Genetic Based Clustering Algorithms and Applications

by

LEE Wing Kin

A dissertation submitted in partial

fulfillment of the requirements for the degree of

Master of Philosophy

in the Division of

Systems Engineering and Engineering Management

of

The Chinese University of Hong Kong

Shatin, N.T.,

Hong Kong SAR, China

July 2000

Genetic Based Clustering Algorithms and Applications

Abstract

Genetic Based Clustering Algorithms and Applications

by

LEE Wing Kin

Clustering methods refer to a group of unsupervised pattern classification procedures that separate or partition a finite collection of objects into subsets based on some predefined criteria. These methods have been applied to many real-life problems. In this thesis, we examine the nature of a clustering problem, develop reliable and efficient algorithms for it, and investigate their practicality in different database applications.

The literature has shown that the clustering of a data array can be stated as a traveling salesman problem (TSP). Hence, TSP structure may be exploited to solve the clustering problem. In this thesis, we explore the use of genetic algorithms (GA) and propose a methodology based on it to solve the clustering problem. In particular, the TSP structure is exploited in our solution methodology. We also consider several clustering problems in information systems. In a typical distributed/parallel database system, a transaction mostly accesses a subset of the entire database. It is, therefore, natural to organize commonly accessed data together and to allocate them into different machine(s)/site(s) in a computer network so as to minimize remote transaction processing. For this reason, data partitioning and data allocation are performance critical issues in distributed database design. In this thesis, we are dealing with data partitioning. In particular, we examine data partitioning in four different contexts: vertical partitioning (VP) a relational database (RDB), horizontal partitioning (HP) a RDB, object-oriented database (OODB) design, and document database design.

To further enhance the performance of our GA clustering algorithms, we propose three new GA crossover operators for solving the TSP. These include a modified ver-

sion of an existing Enhanced Edge Recombination (EER) operator, called Enhanced Cost Edge Recombination operator (ECER), and two new operators, called Shortest Path (SP) operator and Shortest Edge (SE) operator. Their performances are compared with several existing operators using the problem instances from a well-known TSP repository. We run the experiments on a SUN SPARC Ultra-5_10 workstation. Experimental results indicate that the proposed operators have different contributions and that our operators are compared very favourably to others in solving the problem.

論文概述

遺傳分類算法與應用

李永健

分類法是指在某一特定的前題下，把物件集分割成數份或是把相關的物件組成數份，這方法能廣泛地應用在多個領域。在本論文，我們會深入探討分類法的特質，提出一個有效及可靠的分類算法，並研究其數個應用。

在文獻中曾經提及矩陣排列能應用於分類問題上，另外，它能演化成一個旅行商問題或貨郎擔問題(traveling salesman problem，簡稱TSP)。本論文嘗試透過利用TSP結構及了解遺傳算法(genetic algorithms，簡稱 GA)，開發一個新的遺傳分類算法。由於分類算法應用範圍廣泛，我們特別探討所提出的遺傳分類算法如何應用在資訊系統中。在存取一個分散資料庫的過程中，一個交易通常會涉及到存取某一部份的資料庫。如果能夠把相關的資料分類，然後把它們分配在不同的電算機上，資料庫的存取效率也就大大提高。基於上述原因，資料分類和分配這兩個問題在分散資料庫設計上有著重要的影響。在本論文，我們集中研究資料庫的資料分類，並探討我們所提出的新算法於分散資料庫中的四種應用，它們包括垂直分割關聯資料庫、橫切關聯資料庫、對象特性資料庫設計和文件資料庫設計。

另外，我們提出三個新的遺傳雜交算子，包括一個經修改的改良邊重組雜交算子(enhanced cost edge recombination， 簡稱 ECER)， 一個最短路徑算子(shortest path，簡稱SP)和一個最短邊算子(shortest edge，簡稱 SE)，我們運用這三個算子來解決TSP資料庫中的數十個問題。 實驗結果指出，我們所提出的雜交算子於解決TSP問題上有著不同的貢獻， 並且它們比文獻中所提及的數個算子優勝。

Acknowledgments

I am deeply thankful to my supervisor, Professor C. H. Cheng, for the guidance provided. This thesis would not be completed without his insights and direction. Moreover, he gave me valuable suggestions to improve my presentation skills.

I wish to thank the internal reviewers, Professor K. F. Wong and Professor Jeffrey X. Yu, for their comments and suggestions. They, together with my supervisor, chaired the database discussion group from which I got helpful guidance and comments for my research.

I would also like to thank the staff of the Department of Systems Engineering and Engineering Management for their help, and also the Chinese University of Hong Kong for providing a nice environment.

I would like to thank my girlfriend - Miss Chai Kit Yin. She gives me precious private time and provides encouragement when needed. Also, I wish to thank my family members, especially my mother, offers unlimited love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Clustering

Clustering methods refer to a group of unsupervised pattern classification methods that partition the input space into n partitions so as to satisfy some predefined criteria ([43] and [56]). These methods have been applied to many different areas such as manufacturing systems ([1] and [14]), logistics activities ([17], [78] and [97]), information system designs ([87], [93] and [102]).

When cluster partitions are disjoint, one obtains *exclusive* classification (i.e., each data point belongs to only exactly one cluster). On the other hand, when partitions are overlapping, one obtains *non-exclusive* classification [56]. Clustering methods, which seek to provide exclusive classification, can be broadly classified into *intrinsic* and *extrinsic* classification [56]. Intrinsic classification is also called "unsupervised learning" because no category labels are used. Extrinsic classification uses category labels on the objects. For example, suppose that various statistics of personal health have been collected from smokers and non-smokers. An intrinsic classification method groups the individuals based on similarities among the health statistics and then tries to determine whether smoking is a factor in the propensity of individuals toward various diseases. An

extrinsic classification method studies ways of discriminating smokers from non-smokers based on health statistics. In this thesis, we focus on intrinsic classification.

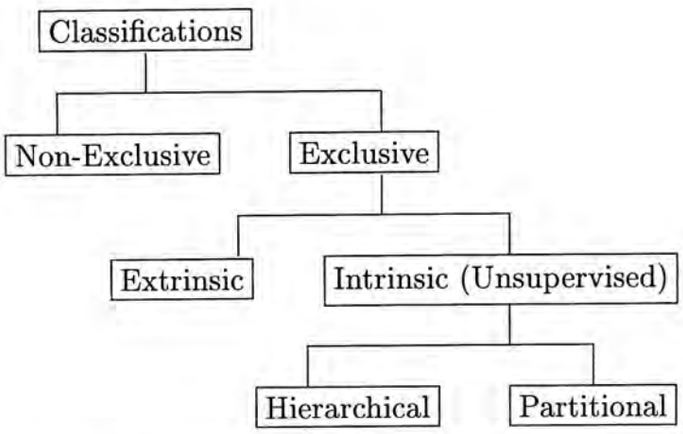

Figure 1.1: Tree of classification types

1.1.1 Hierarchical Classification

Exclusive, intrinsic classification can be subdivided into *hierarchical* and *partitional* [56], (see Figure 1.1). *Hierarchical classification* is a nested sequence of partitions. Suppose n objects to be clustered are included in the set Ψ.

$$\Psi = \{x_1, x_2, \ldots, x_n\}$$

where x_i is the ith object. A partition, Υ of Ψ breaks Ψ into subsets $\{C_1, C_2, \ldots, C_m\}$ satisfying the following:

$$C_i \cap C_j = \Phi \qquad \text{for } i \text{ and } j \text{ from 1 to } m, i \neq j$$
$$C_1 \cup C_2 \ldots \cup C_m = \Psi$$

Notice that Φ is an empty set. A clustering is a partition; the components of the partition are called clusters. For example, let's assume the clustering $\mathcal{C}$ with three

clusters and the clustering $\mathcal{B}$ with five clusters are defined as follows:

$$\mathcal{C} = \{(x_1, x_3, x_5, x_7), (x_2, x_4, x_6, x_8), (x_9, x_{10})\}$$
$$\mathcal{B} = \{(x_1, x_3), (x_5, x_7), (x_2), (x_4, x_6, x_8), (x_9, x_{10})\}$$

$\mathcal{B}$ is nested into $\mathcal{C}$. That means, every component of $\mathcal{B}$ is a proper subset of a component of $\mathcal{C}$. Both $\mathcal{C}$ and $\mathcal{B}$ are clusterings of the set of objects $\{x_1, x_2, \ldots, x_{10}\}$.

Given n objects, there are 2^{n-1}-1 partitions to be considered before they can be divided into two groups [3]. Clearly, when n is large, the number of possible partitions to consider may be very large. To make it more complicated, one has to decide at which level of the hierarchy to stop to determine the number of clusters.

Several popular hierarchical clustering methods have been proposed. They are the single-link, average-link, complete-link, centroid, median and Ward's clustering method (see [56]). Golden and Meehl [37] find that the average-link, complete-link, and Ward's clustering method outperform the others. Besides, Bayne *et al.* [5] conclude that the Ward's method and complete-link method are preferable to median, group average and centroid methods. However, as stated by Anderberg [3], all these methods do not guarantee an optimal solution in terms of the clustering criterion.

1.1.2 Partitional Classification

The problem of *partitional clustering* (non-hierarchical) can be formally stated as follows. Given n patterns in a d-dimensional metric space, determine a partition of the patterns into K groups, or clusters, such that the patterns in a cluster are more similar to each other than to patterns in different clusters. Notice that the value of K may or may not be specified.

The theoretical solution to this partitional problem is straightforward. This is to select a criterion, evaluate if for all possible partitions containing K clusters, and pick the partition that optimizes the criterion. However, as the number of objects increases, the number of possible partitions explodes. For example, there are 34,105 distinct

partitions of 10 objects into four clusters, but this number explodes to approximately 11,259,666,000 if 19 objects are partitioned into four clusters [56]. Clearly, exhaustive enumeration of all possible partitions is not computationally feasible even for small numbers of patterns.

To avoid this combinatorial explosion, several heuristics have been proposed. Many of them start with an initial partition and perform one or more of the following actions: moving objects from one cluster to the other, and merging and splitting clusters. Perhaps the most well-known of these methods is the k-means [56]. It attempts to obtain k cluster centers by minimizing the square-error. It produces partitions which minimizes within-cluster scatter or maximizes between-cluster scatter. However, to avoid local optimum solutions, one has to examine many if not all initial partitions. To generate all possible initial partitions is again not computationally feasible.

Other approaches attempt to eliminate solutions and can be used to achieve an optimal solution. Examples of these are Branch and Bound, integer programming, mathematical programming, etc ([27], [65], [70], [94] and [109]). Although some significant computational savings are realized, these methods are still not computationally feasible for large clustering problems.

1.1.3 Comparative Analysis

Milligan [83], Milligan *et al.* [84] and Milligan and Cooper [85] investigate several clustering methods. They conduct experiments on fifteen clustering algorithms including all the popular hierarchical and partitional approaches. They find that no one single group of algorithms is consistently superior to any other group [83]. For example, the k-means partitional algorithm gives better results than hierarchical methods only when the starting partition is close to the final solution. Besides, Hartigan [44] concludes that different classifications are suitable for different uses. Hence, there is no the best classification. The choice between hierarchical or partitional clustering methods

in fact depends on the domain of the underlying problem.

1.2 Cluster Analysis and Traveling Salesman Problem

Consider the clustering of a non-negative $M \times N$ array. Given two finite sets R and S and a non-negative matrix $(a_{rs})_{r \in R, s \in S}$, where a_{rs} measures the strength of the relationship between elements $r \in R$ and $s \in S$. One would like to permute the rows and columns of the matrix so as to bring its large elements together. The resulting clustering should identify strong relationships between subsets of R and S.

McCormick *et al.* [80] argue that clustering a matrix may be useful for problem decomposition and data reorganization. They illustrate this with three examples. The first one is an *airport design* problem. Given R (S is equal to R; i.e. a similarity array) as a set of 27 facilities that should be available at the airport and are under the control of the designer; a_{rs} is fixed at 0, 1, 2 or 3 depending on whether facilities r and s have no, a weak, a moderate, or a strong dependency. The permuted matrix should suggest a decomposition of the design problem into sub-problems that interact not at all or only in a limited and well-defined ways. The second example involves a set R of 53 *aircraft types* and a set S of 37 functions that they can perform; $a_{rs} = 1$ if aircraft r is suitable for function s, and $a_{rs} = 0$ otherwise. The rearranged matrix shows which aircrafts are able to perform the same functions and which tasks can be performed by the same aircraft. The third example also deals with an object-attribute array. R is a set of 24 marketing techniques, S is a set of 17 marketing applications, $a_{rs} = 1$ if technique r has been successfully used for application s, and $a_{rs} = 0$ otherwise. Lenstra and Rinnooy Kan [72] give a fourth example. It deals with an *input-output matrix*. R (same as S) is a set of 50 regions on the Indonesian islands, $a_{rs} = 1$ if at least 50 tons of rice are annually transported from region r to region s, and $a_{rs} = 0$ otherwise.

As stated by Lenstra [71], all these problems can be modeled as a traveling salesman problem (TSP). TSP is the problem of a salesman who, starting from his home city,

has to find the shortest tour that takes him exactly once through each of a number of other cities and then back to his home city. Suppose there are n cities and c_{ij} is the distance between cities i and j $(i, j = 1, \ldots, n)$. The salesman is interested in a permutation π of {1, ...,n} that minimizes:

$$\sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)}$$

Hence, $\pi(i)$ is the ith city visited. The TSP is symmetric if $c_{ij} = c_{ji}$ for all i, j.

To solve the clustering problem, McCormick *et al.* [80] propose to measure the effectiveness of a clustering by the sum of all products of horizontally or vertically adjacent elements. Notice that higher sums of these products tend to correspond to better clusterings. The problem is now to permute the rows and columns of the matrix so as to maximize this criterion.

Permuting the rows does not affect their horizontal adjacencies of the elements, and permuting the columns does not affect their vertical adjacencies. The problem therefore decomposes into two separate and similar problems, one for the rows and one for the columns. We consider the former. The row optimization problem is to find a permutation ρ of R that maximizes:

$$\sum_{r=1}^{|R|-1} \sum_{s \in S} a_{\rho(r)s} a_{\rho(r+1)s}$$

Here, row $\rho(r)$ of the matrix is put in position r. This is, again, nothing but the symmetric traveling salesman problem [71]. Let $R = \{1, \ldots, |R|\}$, and define

$$n = |R| + 1 \qquad \text{and}$$
$$c_{ij} = -\textstyle\sum_{s \in S} a_{is} a_{js}, \qquad c_{in} = c_{nj} = 0 \qquad \text{for } i, j \in R$$

The rows of the matrix are the cities, the additive inverses of their inner products are the distances, and a dummy city has been added to close the tour. Notice that

column permutation can be done in the same way. Thus, the clustering problem can be reduced to two separate traveling salesman problems. In general, the clustering problem for a p-dimensional array can be stated as p-TSPs, and it may be tackled by an optimal or a heuristic algorithm for the TSP.

The TSP has become the prototypical problem of combinatorial optimization. Moreover, many solution approaches that have become standard in combinatorial optimization are first developed and tested in the context of the TSP. This is partly because its simplicity of statement and its difficulty of a solution are even more apparent than for most other problems in the area [68]. Due to its advantages, many mathematical formulation, applications and solution approaches have been developed [72].

1.3 Solving Clustering Problem

We have shown that clustering is a combinatorial optimization problem which can be reduced to a TSP. It is, therefore, difficult to find an efficient and optimal algorithm that uses polynomial time to solve a given optimization problem. Indeed, the clustering methods described in Section 1.1 can be broadly classified into two categories, namely *exact methods* and *heuristic methods* [30].

Exact methods seek to examine the possibility of an optimal solution. Results obtained by them are the best available. Examples of these are Branch and Bound [65] and dynamic programming [57]. However, if the search space of a problem is very large, these methods may require excessive running time. To avoid examining all the feasible points in the search space, many methods attempt to eliminate solutions. For instance, Branch and Bound approach tries to reduce the complexity through pruning, whereas dynamic programming approach tries to avoid some redundant calculations in the total enumeration. Although some achieve computational savings, most of them are still computationally infeasible for large problems ([9], [30]).

Heuristic methods seek to search for good *approximation* solutions. They are a

popular way of addressing hard problems, because of their simplicity and computational efficiency. Examples of these are the k-means (c-means or basic ISODATA) methods ([54], [55] and [107]). In each iteration of these algorithms, an object is systematically moved to another cluster if such a move reduces the value of the objective function. It is possible that these methods may get stuck at a local minimum. They avoid this problem by taking several different random initial configurations and by applying the procedure to each configuration. This type of evaluation is, however, too *ad hoc* and the quality of the results highly depends on the type of data and objective function ([9],[30]).

Recently, a third class of methods has emerged which are called meta-heuristics or inter-disciplinary approaches. These methods generate new points in the search space by applying operators to current points and statistically move toward more promising areas in the search space. They rely upon an intelligent search of a large but finite solution using statistical methods. These methods do not require to take cost function derivatives and can thus deal with discrete parameters and non-continuous cost functions. They represent processes in nature that are remarkably successful at optimizing natural phenomena [46]. These methods include simulated annealing (SA) [62], tabu search (TS) [62] and genetic algorithms (GA) [36].

SA is inspired by the process that takes place in a crystalline substance during a slow cooling. Starting from a random point in the search space, a random move is made. If this move takes us to a higher point, it is accepted. If it takes us to a lower point, it is accepted only with probability $p(t)$, where t is time. The function $p(t)$ begins close to 1, but gradually reduces towards zero (the analogy being with the cooling of a solid). However, SA only deals with one candidate solution at a time, and so does not build up an overall picture of the search space [6].

As for SA, the TS method is based on gradual local improvement of a current solution of an optimization problem. It searches for a new solution in the neighborhood

of the current one. However, it usually searches the whole neighborhood, instead of picking at random one of its members, as SA does [30].

A third meta-heuristic, the GA, is the subject of this thesis and is described in the following section.

1.4 Genetic Algorithms

John Holland proposed genetic algorithm (GA) [50] in 1975. GA has since become a topic of active research [36] and has been successfully applied in solving some well-known complicated problems such as optimization of gas pipeline [33], Blind Knapsack problem [35], etc. GA is an adaptive method which is based on genetic processes of biological organisms. Over many generations, natural populations evolve according to the principles of natural selection and "survival of the fittest". By mimicking this process, GA is able to "evolve" solutions to real world problems, if it has been suitably encoded. For example, GA can be used to design bridge structures for maximum strength/weight ratio, or to determine the least wasteful layout for cutting shapes from a piece of cloth. It can also be used for online process control, such as in a chemical plant, or load balancing on a multi-processor computer system [6].

In nature, an individual within a population competes with one another for resources (e.g. food or water). Besides, an individual within same the species often competes with others to attract a mate. Those individuals which are most successful in surviving and attracting mates will have relatively larger number of offspring. On the other hand, poorly performing individuals will produce fewer offspring or may even "die out". This means that the genes from the highly adapted, or "fitted" will spread to an increasing number of individuals in each successive generation. In this way, species evolve to become more and more well suited to their environment, see [6].

GA uses a direct analogy of this natural behaviour. It works with a *population* of "individuals", each representing a feasible solution to a given problem. Each individual

is assigned a "fitness score" according to how good a solution to the problem is. The highly fitted individuals are given opportunities to "reproduce", by "cross breeding" with other individuals in the population. This produces new individuals as "offspring" and the least fitted members of the population are less likely to get selected for reproduction, and so "die out". Over many generations, good characteristics are spread throughout the population, being mixed and exchanged with other good characteristics as they go. By favouring the mating of the more fitted individuals, the most promising areas of the search space are explored. If GA is designed well, the population will *converge* to an optimal solution to the problem. The standard GA can be represented as shown in Figure 1.2. For more detail descriptions, see [22], [23], [40], [42], [36] and [82].

```
Begin
    generate initial population
    compute fitness of each individual

    While (NOT finished) Do
    Begin

        For population_size/2 Do
        Begin
            select two individuals from old generation for mating
            recombine the two individuals to give two offspring
            compute fitness of the two offspring
            insert offspring in new generation
        End
        If population has converged Then finished := True

    End
End
```

Figure 1.2: A traditional Genetic Algorithm

Clearly, the large population of solutions and simultaneously searching for better

solutions give the genetic algorithm its power. Indeed, some of the advantages of GA can be summarized as below [46]:

1. Optimizes with continuous or discrete parameters.
2. Does not require derivative information.
3. Simultaneously searches from a wide sampling of the cost surface.
4. Deals with a large number of parameters.
5. Is well suited for parallel computers.
6. Optimizes parameters with extremely complex cost surfaces; they can jump out of a local minimum.
7. Provides a list of optimum parameters, not just a single solution.
8. May encode the parameters so that the optimization is done with the encoded parameters, and
9. Works with numerically generated data, experimental data, or analytical functions.

These advantages are intriguing and could lead to stunning results in situation where traditional optimization approaches fail miserably.

1.5 Outline of Work

In this chapter, we have shown that clustering is a combinatorial problem. By formulate the underlying problem as a TSP, one can take advantage of its problem structure. There are many methods proposed to solve the clustering problem. Most of these methods can be classified into two categories, namely exact methods and heuristic methods. However, a third class of methods called meta-heuristic or inter-disciplinary

approaches has emerged. These methods, including genetic algorithms, have several advantages over the traditional methods. In this thesis, we explore the use of genetic algorithms (GA) and propose a methodology based on GA to solve the clustering problem.

In the following chapter, the TSP literature is reviewed. Also, the applicability of genetic algorithms (GA) to a TSP is presented. To extend the applications of our GA clustering algorithms, we consider several design problems in information systems. They are vertical partitioning (VP), horizontal partitioning (HP), object-oriented database (OODB) design, and document database design.

To evaluate the performance of our proposed algorithm, we compare our approach with the Slagle's, an efficient heuristic method for solving VP problem in Chapter 3. We generate several VP problems whose sizes range from 20 to 100 attributes. Computational results indicate that our proposed algorithm outperforms the Slagle's for the same application.

To further enhance the performance of our GA clustering algorithms, we propose three new operators for solving the TSP in Chapter 4. These include a modified version of the existing Enhanced Edge Recombination operator (EER) [103], called Enhanced Cost Edge Recombination operator (ECER) , two new operators, called Shortest Edge operator (SE) and Shortest Path operator (SP). The performance of the proposed operators are compared with several existing operators using all the problem instances, whose sizes ranges from 14 to 783 cities. The problems are taken from TSPLIB [95]. Experimental results show that the proposed operators have different contributions and that our operators are compared very favourably to others for solving the TSP.

Finally, Chapter 5 outlines the conclusions and future development.

Chapter 2

The Clustering Algorithms and Applications

2.1 Introduction

We have shown that clustering is a combinatorial problem. By formulate the underlying problem as a traveling salesman problem (TSP), one can take advantage of its problem structure. In the following section, the TSP literature is reviewed. Also, the applicability of genetic algorithms (GA) to a TSP is presented. Before a GA can be run, there are several design issues that must be addressed. These include: *initialization, coding, crossover, fitness function, mutation, parent selection, replacement* and *termination.*

To extend the applications of our GA clustering algorithms, we consider several design problems in information systems. They are vertical partitioning (VP), horizontal partitioning (HP), object-oriented database (OODB) design, and document database design. For each application, we review the related literature and formulate a solution model for it. An example is also used to demonstrate the practicality of the proposed algorithm to the underlying application.

2.2 Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a classic combinatorial optimization problem. The problem assumes that a salesman wants to visit N cities with the requirement that each city should be visited by once (except the first city). The distance traveled by the salesman from the starting city to the ending city and from the ending city back to the starting city should be minimized.

Although the structure of TSP is simple, it can be applied to solve many practical problems. For instance, it can be used in designing computer network which connects all computers in a ring topology by cables or optical fibers, ([26] and [99]). Suppose there are N computers and the costs for cabling between computers are known, the problem can be formulated as an N-city TSP. There are many other applications of TSP such as manufacturing circuit board [77], X-ray crystallography [10], VLSI fabrication [64], etc (see also [61], [68] and [72]).

However, solving the TSP is not simple. Given an N-city TSP, it is easy to see that there will be (N-1)! possible tours. For example, if it takes 1×10^{-5} second to evaluate the cost of a tour, it takes 36 seconds to find the optimal tour for a 10-city TSP; and if N grows up to 30, 2.8×10^{18} years would be required to search all combinations! In fact, TSP is well-known to be NP-hard [32].

2.2.1 Related Work on TSP

Traditionally, two approaches, namely exact methods and heuristic methods, are used to solve the TSP. Although the exact methods such as integer programming [81], cutting planes [92], Branch and Bound [69] and dynamic programming [8] can guarantee the solution optimality, it requires excessive computation. On the other hand, heuristic methods such as 2-opt [74], Markov chain [79], nearest insertion [98], farthest insertion, cheapest insertion, sweep, savings etc., are efficient, but they cannot guarantee a good quality solution and many of them do not even have a proven constant worst-case per-

formance ratio [96]. According to Graham *et al.* [38], the best known performance ratio for solving the TSP is obtained by Christofides' heuristic [16]. It obtains a performance ratio of at most $\frac{3}{2}$. That means, the worst-case obtained by it is guaranteed to be less than $\frac{3}{2}$ times the optimal solution.

A third class of solution methods has emerged in recent years, which is called meta-heuristic methods or inter-disciplinary approaches. These include simulated annealing (SA) [62], genetic algorithms (GA) [36], and tabu search (TS) [47]. But Bhuyan *et al.* [9] claim that the disadvantages of SA in solving combinatorial optimization problems are that a tremendous amount of execution time is needed and the determination of an efficient annealing schedule is difficult. Also, SA only deals with one candidate solution at a time, and so does not build up an overall picture of the search space [6]. Knox [63] reports that TS and a modified version of simulated annealing [67] exhibit similar performance for solving the TSP. Homaifar *et al.* (1993) [51] state that if a GA is well designed, it can be comparative with the best known techniques including the SA and TS. In fact, GA has shown great promise in solving some very complicated combinatorial problems including some large-scale TSPs and produced significant improvements in this area ([11], [34], [39], [51], [60], [91], [103], [108] and [112]). GA is suitable for the problem because it quickly directs a search to promising areas of the search space. In this thesis, we use genetic algorithms to solve the TSP and investigate the practicality of GA for information systems applications.

2.2.2 Solving TSP using Genetic Algorithm

Before a GA can be run, a suitable *coding* or *representation* for the problem must be devised. We also require a *fitness function*, which assigns a figure of merit to each coded solution. During the run, parents must be *selected* for reproduction, and *recombined* to generate offspring.

Notice that each of the above components alone can be regarded as a research

topic. In fact, different researches have tried to enhance the performance of GA for solving the TSP by proposing various techniques in each of the above components. For instance, Tamaki *et al.* [106] propose a new method for coding a TSP and find that the search for the optimal tour is more effective. Whitley [111] investigates the parent selection scheme and develops a new rank based selection method to obtain better results. As stated by Falkenauer [30], perhaps the most important technique in GA is the crossover, also called the recombination operator. Indeed, several operators have been proposed for TSP [82]. In this section, we first address various design issues in GA for solving TSP: *initialization, coding, crossover, fitness function, mutation, parent selection, replacement* and *termination*. In Chapter 4, we propose three new crossover operators that can further enhance the performance.

Initialization

Initialization involves generating initial solution to the problem. The initial solutions can be generated randomly or using some heuristic methods. For simplicity and to avoid additional overheads, we generate the initial population randomly.

Coding

Traditionally, GA used binary representation, e.g. $x_1 = (01011001)$, which is often termed chromosome. However, since each digit has cardinality of 2, higher cardinality alphabets have been used and some researchers claim that it has advantages over the traditional coding [6]. In our implementation, we use non-binary representation. Several non-binary coding methods were proposed such as adjacency, ordinal, path and ordered [82]. For traveling salesman problem, the most natural representation of a tour is the sequence of cities in the route, i.e. the i-th number represents the city which must be visited on the i-th position in the order. For instance, $x_1 = (4, 3, 2, 1, 5)$ gives the sequences of cities in a solution of a 5-cities TSP.

Crossover

Crossover requires two individuals to exchange their genetic composition. The offspring then inherits some genes from its parents via this operation. Traditional GA uses 1-point crossover. When a crossover operator is to be carried out over a pair of parents, a cutting point will be usually selected randomly, and the chromosomes of the parents will be both split at that point and then the segments of those chromosomes will be exchanged to give birth to the offspring. However, many different crossover algorithms have been devised, and many involve more than one cutting point. An advantage of having more crossover points is that the problem space may be searched more thoroughly. In fact, DeJong [24] investigates the effectiveness of multiple-point crossover, and concludes that 2-point crossover leads to performance improvement, further adding crossover points have adverse effect. The problem with further adding crossover points is that building blocks are more likely to be disrupted.

Several operators have been proposed for TSP: Partially-mapped (PMX) (Goldberg and Lingle [34]), Order (OX) (Davis [21]), Cycle (CX) (Oliver *et al.* [91]), OX2 and Position Based (PB) (G. Syswerda [105]), Edge Recombination (ER) (Whitley *et al.* [112]), Enhanced Edge Recombination (Starkweather *et al.* [103]). The above operators are 2-point crossover except ER and EER.

Take the Order operator as an example. The offspring inherits the elements between the two crossover points from the selected parent in the same order and positions as they appear in that parent. The remaining elements are inherited from the other parent in the order they appear, beginning with the first position following the second crossover point and skipping over all elements already present in the offspring.

Parent 1:	a b c d e f g h i
Parent 2:	c f a h d i b g e
Crossover point:	* *
Offspring:	f g a h d i b c e

Figure 2.1: An example of order crossover operator

An example is given in Figure 2.1. The elements a, h, d, i and b are inherited from Parent 2 in the order and position in which they occur in Parent 2. Then starting from the first position after the second crossover point, the offspring inherits from Parent 1. In this example, position 8 is the next position. In Parent 1, h is located in position 8. However, it is already present in the offspring, so Parent 1 is searched until an element is found which is not already present in the offspring. In this case, c is inherited from Parent 1. This process continues until the offspring is complete.

Starkweather *et al.* [103] study several operators and conclude that for the TSP the important information would seem to be the adjacency information. The ER operator explicitly preserves adjacency information and clearly has the best performance on this problem. Moreover, the enhanced ER (EER) further improve the performance of the system [103]. In Chapter 4, three new operators are proposed for TSP. The performances of them together with the above operators are evaluated, results are shown in Section 4.5.

Fitness Function

A fitness function must be used to evaluate the "fitness" (value) of the individuals within the population. Parents are selected from the population using a scheme that favours the more fitted individuals to produce offspring. Good individuals will probably have more opportunities to be selected as parents and poor ones may not be at all.

Obviously, the fitness function used in TSP is simply the total distance traveled

by the salesman. Therefore, more fitted individuals will have smaller total distance traveled. Since the distance between any two cities are known, the total distance of the route can be obtained easily for each solution (individual).

Mutation

Mutation is applied to each child individually after crossover. It randomly alters each gene with a small probability (typically 0.001). Mutation, thus, provides a small amount of random search, and helps to ensure that no point in the search space has a zero probability of being examined.

Several mutation operators are suggested [82]. For example, Oliver *et al.* [91] develop a mutation operator called SWAP which randomly swap two cities in a TSP sequence. However, the usage and design of different operators are still immature and face a number of open issues. For instances, Nürnberg *et al.* [89] evaluate several mutation operators and conclude that there can be a trade-off between good convergence rates and reachable solutions depending on the mutation operators used. However, the choice of appropriate mutation operators may need to be done by some kind of empirical data analysis drawn from the actual evolutionary dynamics or even by a self-adaptive process [31]. Obviously, it is not an easy task. In addition, some crossover operators such as ER and EER already provide an effective mutation rate [112]. Therefore, in order to provide a fair comparison of different crossover operators, we do not consider any mutation operator in our implementation. An analysis of the choice of mutation operators remains as a future task.

Parent Selection

Parent selection is a process that allocates reproductive opportunities to individuals. The biased selection enables the convergence of the search. As the population converges, so the range of fitness in the population reduces. However, this sometimes leads to

premature convergence and *slow finishing*.

Premature convergence means that the genes from a few comparatively highly fit (but not optimal) individuals may rapidly come to dominate the population, causing it to converge on a local optimum. *Slow finishing* is the reverse problem to premature convergence. After many generations, the population will have largely converged, but may still not have precisely located the global optimum. The average fitness will be high, and there may be little difference between the best and the average individuals. Consequently there is an insufficient gradient in the fitness function to push the GA towards the global optimal solution.

There are many methods to overcome these problems. Several are described in [4]. The commonly employed methods include *fitness scaling* and *fitness ranking*. *Fitness ranking* overcomes the reliance on an extreme individual. Individuals are sorted in order of raw fitness, and then reproductive values are assigned according to rank. In *fitness scaling*, the maximum number of reproductive trials allocated to an individual is set to a certain value. Whitley [111] conducts some experiments and shows that fitness ranking to be superior to fitness scaling.

To allocate reproductive trials to individual so that higher ranked individual will obtain higher reproductive trials, it can be done linearly or exponentially. In our implementation, we adopt the approach of Whitley [111] to rank individuals according to their fitness and in producing *selective pressure* (bias), i.e. a linear function is used. For example, the 5th ranked individual will end up with the same chance to be selected as its parent each time regardless of its fitness value of those above (or below). Also, for selective pressure equals to 1.5 implies that the top ranked individual in the population is 1.5 times more likely to reproduce (in one reproductive cycle) than the median individual in the population.

Replacement

There are two replacement approaches, named generation gap and steady-state replacement. The generation gap is defined as the proportion of individuals in the population which are replaced in each generation. Most work has used a generation gap of 1, i.e. the whole population is replaced in each generation [40]. However, a more recent trend has favoured steady-state replacement ([110], [111] and [104]). It replaces a few individuals in each generation.

In our implementation, we adopt a steady-state approach similar to that of GENITOR [111], i.e. in each generation only two worst individuals are replaced. Therefore, parents and offspring may co-exist in a population.

Termination

The process of crossover and replacement are repeated until the population converges or attains a pre-specified maximum number of generations. In our implementation, we employ the former criterion. The population is said to have converged when all of the genes (individuals) have the same fitness value. As the population converges, the average fitness will approach that of the best individual. In our implementation, we set the termination until population converges.

Implementation

The pseudo-code of the algorithm is given in Figure 2.2.

Input cost matrix for TSP
Generate an initial population of N random solutions (individuals)

While (termination criterion not satisfied) **do**

Select two parents P_1 and P_2 according to fitness ranking
Using the crossover operator to generate two new offspring
Replace two worst individuals in the population with two new offspring

End while

Output top ranked solution

Figure 2.2: Genetic algorithm for solving the TSP

In Figure 2.2, the input to the algorithm is the cost matrix of the TSP. The algorithm starts by generating N number of individuals. For each generation, two parents are selected to generate two new offspring. Two worst individuals will be replaced by these two new offspring in the population. The above process terminates when the stopping criterion is reached and the algorithm output the top ranked individual.

We implemented the algorithm using C++ and the program ran on a Sun SPARC Ultra-5_10 workstation. In the next section, we use the proposed algorithm to solve several design problems in information systems.

2.3 Applications

2.3.1 Clustering for Vertical Partitioning Design

To cope with the increasing volume of today database applications, cluster computing technology provides an efficient solution. Cluster computers facilitate high performance and high reliability. These features are paramount in large database applications. Using them as database servers, data are typically partitioned and distributed widely over the cluster. This effectively results in a classical distributed or parallel database

environment (see [2] and [90]), respectively.

Response time in a distributed or parallel [1] database system is largely determined by how the database programs and data are organized and stored on the different machines/sites. In practice, it is quite common that the database programs are available on each site and the main design issue in distributed databases reduces to the distribution of data. The concept is to place related data (e.g. a frequently accessed group of attributes) on near-by, preferably on the same, sites. Nevertheless, identification of such groups is not straightforward especially in large applications involving thousands of database transactions. In general, the study of the data distribution requires solving two problems: the partitioning problem and the allocation problem [115]. In this thesis, we deal with the partitioning problem and propose a genetic search based algorithm to solve it.

Given the work-profile of a database application, in terms of the transactions and the data they access, the objective of our algorithm is to identify the aforesaid data groups. Individual groups, commonly known as database fragments, could then be placed on the most appropriate computer site(s). Without loss of generality, our algorithm was designed under the following assumptions:

1. The relational database model is assumed [20]. Today, relational databases are by far the most wide-used in the industry.

2. The application work-profile, i.e. the frequently accessed data and the corresponding transactions, is known in advance. Based on this information, user access patterns can be estimated, and from it, important database transactions could be located.

[1] To simplify the discussion, we focus on distributed database design. But the proposed method is equally applicable to parallel database design.

3. In vertical partitioning (see the next paragraph) applications, it is assumed that the primary key of a relation is duplicated in every vertical fragments produced. In this way, the reconstruction of the whole relation from its vertical fragments is possible through the join operation.

A relation is essentially a table. Dividing a table into smaller ones requires two elementary operations: vertical partitioning and horizontal partitioning. Consider a relation PROJECT concerning all ongoing projects of a company in Figure 2.3. We may horizontally divide it into two smaller units in Figure 2.4. The primary key ProjNo is duplicated in both relations so that the original relation can be re-constructed. Similarly, we may vertically divide it into two smaller units, as shown in Figure 2.5. Vertical and horizontal partitioning are elementary operations. They can also be nested and leading to hybrid fragments.

PROJECT

ProjNo	ProjName	Budget	Location
J1	Database development	130000	Michigan
J2	Group technology	115000	Illinois
J3	CAD/CAM	240000	Michigan
J4	Maintenance	330000	Iowa

Figure 2.3: The PROJECT relation

PROJECT 1

ProjNo	ProjName	Budget	Location
J1	Database development	130000	Michigan
J2	Group technology	115000	Illinois

PROJECT 2

ProjNo	ProjName	Budget	Location
J3	CAD/CAM	240000	Michigan
J4	Maintenance	330000	Iowa

Figure 2.4: Examples of horizontal partitions

PROJECT 3

ProjNo	ProjName	Location
J1	Database development	Michigan
J2	Group technology	Illinois
J3	CAD/CAM	Michigan
J4	Maintenance	Iowa

PROJECT 4

ProjNo	Budget
J1	130000
J2	115000
J3	240000
J4	330000

Figure 2.5: Examples of vertical partitions

Notice that relation fragments PROJECT1 and PROJECT2 (see Figure 2.4) are the tuple sets J1, J2 and J2, J3, respectively. Similarly, the same for PROJECT3 and PROJECT4 (see Figure 2.5) are the tables defined by [ProjNo, ProjName, Location] and [ProjNo, Budget], respectively.

These fragments are not randomly formed. It is the role of the database designers to determine how best to partition the original relation in order to achieve the highest performance and/or reliability over a computer cluster.

Related Work for Vertical Partitioning Problem

The motivation of vertical partitioning (VP) in database design is to minimize the number of page accesses while create smaller fragments to satisfy user queries. As Navathe *et al.* [87] point out, if a relation has n non-primary key attributes, the number of possible fragments to consider will equal to n^{th} Bell number, $B(n)$. For large value of n, $B(n) \approx n^n$. For example, when $n = 10$, $B(n) \approx 115,000$; when $n = 15$, $B(n) \approx 10^9$; and when $n = 30$, $B(n) \approx 10^{23}$. From this scale, it is not difficult for one to appreciate the complexity of the vertical partitioning problem.

Hoffer [49] has formulated a 0-1 nonlinear integer-programming model for the vertical partitioning problem. The model minimizes storage, retrieval, and update costs which subject to the capacity constraints on database sub-files. An approximate solution based on the bond energy (BE) algorithm is used. Eisner and Severance [28] propose to identify the most frequently accessed data fragments and place them in high-speed primary memory. This partitioning problem is isomorphic to the minicut-maxflow network problem, which can be solved by the Ford/Fulkerson algorithm. However, the solution method is inefficient for large problems. Hammer and Niamir [45] design a mechanism that can find a near optimal vertical partition, although it conducts a search through the space of all possible partitions by employing the hill-climbing technique.

Navathe *et al.* [87] extend the work of Hoffer. Affinity among attributes is defined to express the extent to which they are simultaneously processed. The BE (Bond Energy) algorithm is introduced which partitions attributes according to their affinity. Since the BE algorithm does not necessarily produce a solution in a diagonal structure, a heuristic algorithm is required to divide attributes into overlapping or non-overlapping fragments. Cornell and Yu ([18], [19]) develop an integer programming formulation to solve the problem of vertical partitioning. At each iteration, the integer programming formulation finds an optimal partitioning that splits the relation into two fragments. The integer programming formulation can be applied recursively until no profitable split can be found. However, this approach only finds a local optimal partition. Navathe *et al.* [88] propose an algorithm for vertical partitioning in 1989, which uses a graphical technique. The major feature of this algorithm is that all fragments are generated by one iteration in a time of $O(n^2)$. However, as pointed out by Lin and Zhang [76], it has some undesirable features.

Later, Lin *et al.* [76] propose a new cluster model and a graphical vertical partitioning algorithm to overcome some deficiencies found in the algorithm of [88]. It

has proved to be more efficient that the algorithm in [87] and more effective than [88]. Cheng [13] propose a new vertical partitioning algorithm based on a branch and bound approach. In a binary access matrix, the algorithm outperforms the BE algorithms used by Hoffer and Navathe *et al.*. Huang and Van [52] propose an heuristic search algorithm to search the large solution space of partitions and to choose one partition that yields the minimum number of disk accesses by using the A^* technique.

Refer to Section 1.3, these clustering methods can be broadly classified into two categories, namely *exact methods* and *heuristic methods.* Most of them are either so computational cumbersome which actually cannot be applied to solving any practical problem of moderate size, or are efficient but offer no guarantee to find any solution of reasonably good quality. A third class of methods has emerged recently. These methods include simulated annealing (SA) [62], tabu search (TS) [47] and genetic algorithms (GA) [36]. These new methods have shown great promise in solving some very complicated combinatorial problems in finite computation time.

In this thesis, we focus on genetic algorithms and propose a methodology based on genetic algorithms for database application design. In the next section, we present the model of vertical partitioning design. Due to the advantages of the traveling salesman problem (TSP) (see Section 1.2), we formulate the vertical partitioning problem as a TSP. Then, we present an example, which is adopted from a literature, to demonstrate the practicality of GA in solving the vertical partitioning problem. Several problems are generated in this problem and the performance of it is compared with a well-known algorithm in Chapter 3.

The Model

An application work-profile describes the access patterns of a set of transactions {T1, T2, ...}, say, over the attributes of the database relation, i.e. {A1, A2, ...}. For design purpose, it is commonly modeled by a transaction-attribute matrix. Consider the

transaction-attribute matrix (Figure 2.6). It contains five non-primary key attributes, i.e. {A1, A2, A3, A4, A5}, and four transactions, i.e. {T1, T2, T3, T4}, accessing the relation. A "1" (or "0") entry in the matrix indicates that the corresponding transaction uses (or does not use) the attribute(s) concerned. To demonstrate the clustering concept, we assume that the access of frequencies of all transactions are the same. This assumption will be relaxed later. Notice that in Figure 2.6, the distribution of "1" in the matrix is completely random. Such a matrix is highly inefficient in database design.

	Attributes					
	A1	A2	A3	A4	A5	Access Frequencies
Transactions						
T1		1		1	1	20
T2	1		1			20
T3		1		1		20
T4	1		1			20

Figure 2.6: Transaction-attribute matrix 1

Consider another transaction-attribute matrix 2, (Figure 2.7). It is formed by rearranging certain rows and columns of matrix 1. Matrix 2 comprises a diagonal block structure. TC-1 is a transaction cluster which accesses AC-1, an attribute cluster; similarly, TC-2 accesses AC-2.

TC-1/AC-1 and TC-2/AC-2 form two perfectly separable sub-matrices. Practically, a transaction in a sub-matrix only accesses attributes in the same sub-matrix. Attributes in a sub-matrix make up a fragment. These fragments could then be distributed over the computer system. This lays down the objective of vertical partitioning. In distributed database design, the designers decompose the corresponding transaction-

Transactions		Attributes: AC-1 A1	AC-1 A3	AC-2 A5	AC-2 A2	AC-2 A4	Access Frequencies
TC-1	T2	1	1				20
	T4	1	1				20
TC-2	T3				1	1	20
	T1			1	1	1	20

Figure 2.7: Re-arranged transaction-attribute matrix 2

attribute matrix into sub-matrices. Attributes in a sub-matrix should closely match the data access requirements of the transactions in the same sub-matrix. The goal of vertical partitioning is to maximize the number of "1" entries retained in sub-matrices. But clearly separable sub-matrix patterns are not easy to determine; especially in real life situations, e.g. banking, where the number of transaction could be thousands and the number of attributes could be up to hundreds. For instance, consider the transaction-attribute matrix 3 (see Figure 2.8).

Transactions		Attributes: AC-1 A2	AC-1 A5	AC-2 A3	AC-2 A1	AC-3 A4	AC-3 A6	Access Frequencies
TC-1	T1	1	1			1		20
	T2	1	1				1	20
TC-2	T3			1	1	1	1	20
	T4			1	1			20

Figure 2.8: Another transaction-attribute matrix 3

Completely separable sub-matrices do not exist in matrix 3 because attributes A4 and A6 are accessed by transactions from different sub-matrices. Attributes 4 and 6 are known as inter-sub-matrix attributes. Although, these inter-sub-matrix attributes are common and prevent the formation of clearly separable diagonal sub-matrices, existing clustering algorithms cannot handle them effectively, e.g. [87].

In this example, notice that the removal of inter-sub-matrix attributes A4 and A6 would lead to the formation of two perfectly separable sub-matrices. In general, to deal with the inter-sub-matrix attributes, e.g. A4 and A6 in matrix 3, the following two options may be taken in distributed database design.

1. Duplicate the inter-sub-matrix attributes into all of the identified sub-matrices.

2. Create an additional sub-matrix comprising the inter-sub-matrix attributes.

The choice of these design options depends on the update frequency of the transactions. The former requires update of multiple copies of the same data. This inevitably would undermine the performance of the system. However, if updates are infrequent, multiple updates would be affordable. Under this circumstance, option 1 would be preferred. On the other hand, if update transactions are frequent, option 2 would be a better choice as data would be isolated and data inconsistency would be avoided.

The TSP Solution

The problem of determining a desirable permutation for rows and columns in a solution matrix can be formulated as a Traveling Salesman Problem (TSP) [72]. To set up the TSP-VP problem, we will make use of distance measures between a pair of rows (attributes) and columns (transactions).

We define the distance (cost) between transaction T_i and T_j as follows:

$$d_{ij} = \begin{cases} \sum_{k=1}^{n} |a_{ik} \times F_i - a_{jk} \times F_j| & \text{if} \quad a_{ik} \neq a_{jk} \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where a_{ik} is the entry in the unorganized transaction-attribute matrix (i represents transaction T_i and k represents attribute A_k) and F_i is the access frequencies for transaction T_i per unit time period (e.g., a day).

Similarly, we use the following distance (cost) measure for attributes A_i and A_j.

$$d_{ij} = \sum_{k=1}^{m} |(a_{ki} - a_{kj}) \times F_k| \tag{2.2}$$

In TSP, the total distance is calculated as the distance traveled by the salesman from the starting city to the last city plus the distance from the last city back to the starting city. The TSP objective is to minimize the total distance traveled by the salesman provided that each city (except the starting city) should be visited by once. In the VP, the first and last attributes/transactions need not be connected and we observe that there is no constraint to govern the selection of the starting attribute/transaction. For this reason, we introduce a dummy attribute/transaction in our VP with distance zero for connecting to every attribute/transaction, we can then formulate the problem as a TSP.

Using the example in Figure 2.6, we compute the distance measure between attributes using Equation 2.2. We obtain the following cost matrix:

	A_1	A_2	A_3	A_4	A_5
A_1	0	80	0	80	60
A_2	80	0	80	0	20
A_3	0	80	0	80	60
A_4	80	0	80	0	20
A_5	60	20	60	20	0

Figure 2.9: Cost matrix for attributes of the example in Figure 2.6

Similarly, using Equation 2.1, the cost matrix for transactions is given as:

	T_1	T_2	T_3	T_4
T_1	0	100	20	100
T_2	100	0	80	0
T_3	20	80	0	80
T_4	100	0	80	0

Figure 2.10: Cost matrix for transactions of the example in Figure 2.6

After we obtain the cost matrix for attributes/transactions, we can introduce a dummy attribute/transaction into the problem. For instance, two dummy variables: attribute A6 and transaction T5 are introduced into the cost matrix with distance zero for connecting to every attribute/transaction. By solving the permutation problem as a TSP, we obtain the following tours for attribute and transaction grouping:

A1-A3-A5-A2-A4-A6	Total cost: 80
T3-T1-T5-T2-T4	Total cost: 100

By removing the dummy attribute/transaction, we have two sequences: {A1, A3, A5, A2, A4} and {T2, T4, T3, T1}. Note that we obtain the re-arranged transaction-attribute matrix as shown in Figure 2.7.

Clearly, two fragments: AC-1 and AC-2, can be easily identified in Figure 2.7. However, in real life situations such as banking, where the number of attributes could be hundreds, clearly sub-matrix patterns are not easy to determine. In these cases, a splitting algorithm can be applied to further cluster the solution sequence. Anderberg (1973) [3] discusses seven hierarchical clustering techniques. Among the seven techniques, single linkage, average linkage and complete linkage clustering are most widely used. For instance, in the above example, we can cut the attribute sequence at the edge A3-A5 to obtain the two fragments. It is because the edge A3-A5 contributes the highest cost in the solution. Note that this method is similar to the single linkage clustering method.

There are several approaches that have been proposed for the permutation of attributes. A well-known approach is the Bond Energy Algorithm (BE) proposed by McCormick *et al.* [80]. Slagle *et al.* [100] modify the BE and use it in data-reorganization. In the next section, the applicability of GA to TSP-VP is described by using an example from the literature. By formulated the VP problem as a TSP, VP is achieved when the associated TSP is solved. In Chapter 3, we compare the performance of our GA clustering algorithm with the Slagle's.

An Example

An example with 20 attributes and 15 transactions is adopted from Navathe *et al.* [87] to provide a comprehensive understanding of our algorithm in solving the VP. The performance of the GA depends on many factors: (1) population size, (2) termination criteria, (3) selective pressure. After tuning the parameter values through several experiments , we set the population size to 1200 and selective pressure to 1.2 (see Table 4.4). Enhanced Cost Edge Recombination (ECER) operator (see Chapter 4) is used and the GA is run until the population converges. We implemented the algorithm using C++ and the program ran on a Sun SPARC Ultra-5_10 workstation.

The computation time is about 10 seconds and the total number of trials (generations) for attribute grouping is 3017 and for transaction grouping is 8774. All solutions converge. They have the same total distance of 720 for attribute grouping and 875 for transaction grouping. (Note that we are only interested in grouping attributes. Transactions are permuted in order to obtain a better understanding of the re-arranged transaction-attribute matrix.) Figure 2.11 shows the re-arranged transaction-attribute matrix after permutation of attributes and transactions.

	Attribute																			
	14	2	12	13	9	3	7	10	11	17	18	16	15	20	19	4	6	5	1	8	
Transactions																					Access Freq.
T3						1	1	1	1	1	1										50
T7						1	1	1	1	1											15
T12	1						1	1						1	1	1					10
T9	1	1							1						1			1			10
T6								1	1										1	1	15
T5																		1	1	1	15
T10					1						1	1							1		10
T13									1	1	1	1	1							1	10
T15										1	1	1	1	1	1						5
T4												1	1	1	1						50
T8	1	1	1	1							1	1	1	1							15
T2	1	1	1	1	1																50
T11		1	1	1	1	1											1	1	1		10
T14		1			1	1	1									1	1	1	1	1	5
T1																1	1	1	1	1	50

Figure 2.11: An example for solving the VP

To obtain fragments, we cut the attribute tour at the edge having the highest cost. (Note that this method is similar to the single linkage method, see [3].) Such cutting is reasonable since it meets the subjective criterion [87] of a 'good' vertical partitioning, that is,

(I) attributes most frequently accessed together by transactions should form a fragment; and

(II) all pairs of attributes in a fragment have high affinity 'within fragment' but low affinity 'between fragments'

However, this approach might not determine the best clustering result because the distances between attributes within each fragment are not taken into consideration. To determine the best clustering result, the selection criterion developed by Stanfel [102] can be used. (Note that this method is similar to the average linkage method, see [3].) This selection criterion seeks to minimize the average distance within groups and maximize the average distance between groups. First, define:

$$Y_{ij} = \begin{cases} 1, & \text{if records } i \text{ and } j \text{ are in the same group} \\ 0, & \text{otherwise} \end{cases}$$

The expression for the average distance within groups is given as

$$\frac{\sum_{i=1}^{M-1}\sum_{j=i}^{M} d_{ij}Y_{ij}}{\sum_{i=1}^{M-1}\sum_{j=i}^{M} Y_{ij}} \tag{2.3}$$

where d_{ij} is the distance between the attributes i and j.

While the expression for the average distance between groups is given as

$$\frac{\sum_{i=1}^{M-1}\sum_{j=i}^{M} d_{ij}(1-Y_{ij})}{\sum_{i=1}^{M-1}\sum_{j=i}^{M}(1-Y_{ij})} \tag{2.4}$$

Hence, in order to achieve the objective of maximizing the homogeneity of records within groups as well as the heterogeneity of records between groups, the difference between the average distance within groups and the average distance between groups is minimized as shown in Equation 2.5:

$$\frac{\sum_{i=1}^{M-1}\sum_{j=i}^{M} d_{ij}Y_{ij}}{\sum_{i=1}^{M-1}\sum_{j=i}^{M} Y_{ij}} - \frac{\sum_{i=1}^{M-1}\sum_{j=i}^{M} d_{ij}(1-Y_{ij})}{\sum_{i=1}^{M-1}\sum_{j=i}^{M}(1-Y_{ij})} \tag{2.5}$$

The partition point is the result that gives the minimum value of Equation 2.5. This clustering method is a sequence of partitions in which each partition is nested into next partition in the sequence and is known as hierarchical clustering method (see Section 1.1.1).

Clearly, the objective value of Equation 2.5 can be used as statistics to compare cluster validity. It measures the degree of linear correspondence between attributes within the same group. Small values imply that the attributes within the same group agree to each other. In fact, this statistic is quite similar to Hubert's Γ statistics [53], which can be used to test cluster validity.

The above approach is used to identify fragments. Four fragments are identified, which are: {14, 2, 12, 13, 9}, {3, 7, 10, 11, 17, 18}, {16, 15, 20, 19} and {4, 6, 5, 1, 8}.

Note that the fragments identified are the same as the one obtained by Navathe *et al.* [87].

Conclusions

In this section, we present the vertical partitioning (VP) problem. We then formulate the problem as a traveling salesman problem (TSP) and propose a methodology based on genetic algorithms (GA) to solve the corresponding TSP-VP. We also adopt an example from a literature to demonstrate the practicality of our new GA clustering algorithm in solving the VP problem.

2.3.2 Horizontal Partitioning a Relational Database

As we have discussed earlier (see Section 2.3.1), horizontal partitioning (HP) decomposes a relational table along its tuples. There are two related but different types of partitioning: *primary* and *derived.* Primary horizontal partitioning of a relation is performed using predicates that are defined on that relation. Derived horizontal partitioning is the partitioning of a relation which results in predicates being defined on another relation. Our discussion on horizontal partitioning mainly focuses on the former.

Given a relation PROJECT, the SQL query "**SELECT** * **FROM** PROJECT **WHERE** Budget > 140000" will extract the information of all projects with budgets over $140,000. At this point, we are only interested in the simple predicate (i.e. Budget > 140000). The predicate forms the basis of one of the horizontal partitions in Figure 2.4. In general, given a relation R($A_1, A_2, \ldots, A_n$), where A_i is an attribute defined over domain D_i, a simple predicate p_j defined on R has the form p_j: $\{A_i \theta \text{Value}\}$ where $\theta \in \{=, <, \leq, \neq, >, \geq\}$ and Value $\in D_i$. SQL queries containing simple predicates are common. But in some occasions, e.g. in decision support system (DSS) applications, *ad hoc* queries are also frequent. These queries inevitably involve complicated predicates.

A complicated predicate consisting of multiple simple predicates[2] could be decomposed into an equivalent collection of SQL queries with simple predicates. For this reason, irrespective of the complexity of the predicates, horizontal partitioning could be achieved by analyzing them.

Primary horizontal partitioning: Given a SQL query containing a set of simple predicates accessing a relation R, we define P as the set of all simple predicates, $P = \{p_1, p_2, \ldots, p_m\}$. The conjunction of these predicates (or some negatives) forms the fragment schema on the relation R. The objective function of primary horizontal partitioning is to define some complicated predicates (involving logical OR and/or AND operations) to decompose the relation R into fragments which will then be distributed over the computer clusters.

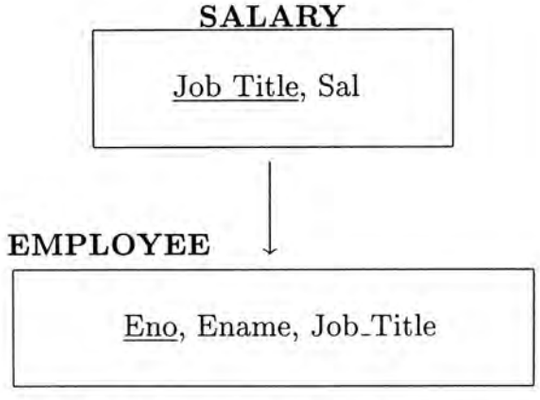

Figure 2.12: Connection of relations using a link

Derived horizontal partitioning: Database transactions often access more than one relation. Related database relations can be connected through joins. For example, in Figure 2.12, relations SALARY and EMPLOYEE are connected by a link, L, representing a join. The member of the link is EMPLOYEE and the owner of the link is SALARY.

[2]In disjunctive normal form, multiple conjunctive predicates are connected with disjunction operators. Similarly, in conjunctive normal form, multiple disjunctive predicates are connected with conjunctive operators.

Horizontal partitions of the member relation of a link are derived using a selection operation specified on the owner relation of the link. In other words, the member relation will be partitioned according to the partitioning of the owner relation. The partitioning algorithm is similar to the partitioning algorithm for primary horizontal partitioning except the partitions from the owner relation are used.

Related Work for Horizontal Partitioning Problem

Ceri and Pelagate [12] develop an iterative procedure for horizontal partitioning. The set P of predicates is complete if and only if any two tuples belonging to the same partition are referenced with the same probability by any database applications. The set P is relevant if and only if each predicate in the set partitions the relation at hand. A simple predicate is relevant in determining a partition if and only if the predicate and its negation are referenced differently at least by one application. Ceri/Pelagate's partitioning algorithm involves adding relevant predicates to the set P until it is complete. It then forms database fragments based on P. Their algorithm is shown in Figure 2.13.

1 Find one relevant predicate p_1.

2 Let $P = \{p_1\}$.

3 Find another relevant predicate and add this to P.

4 Continue until P is complete.

5 Form the fragment schema for this relation.

Figure 2.13: Ceri and Pelagate's algorithm for horizontal partitioning

They further impose a frequency constraint on each resulting fragment. This constraint ensures that the probability of access to each tuple in a candidate fragment must be the same. If this property does not hold, then further horizontal fragmenta-

tion may take place; that is, the fragment has not been fully reduced. As pointed out by Zhang and Orlowska [114], the test of completeness involves the comparison of the probabilities of access by any applications. Thus, as the size of set P increases, the cost of calculation can be very expensive.

Alternatively, Zhang and Orlowska [114] define predicate affinity. The BE (i.e. Bond Energy) algorithm [80] is then used to cluster predicates, and a horizontal partition is formed for each cluster. However, the BE algorithm may not necessarily produce clusters along the diagonal during cluster identification. When this happens, cluster extraction will require additional computation.

An Example

Similar to the transaction-attribute matrix for vertical partitioning, we use a transaction-predicate matrix to represent the predicate access pattern of a set of given transactions. This matrix is used to define the primary horizontal partitioning model. Based on the transaction-predicate matrix, we identify and remove inter-cluster predicates in order to form clusters of transactions using the same set of predicates. Further, using the clusters and inter-cluster predicates, we can define the database partitions (i.e. fragments). We use the example of Zhang and Orlowska [114] to illustrate our partitioning strategy. In the example, R = (Eno, Ename, Sal, Degree) is a relation schema and there are seven transactions (T1-T7) using nine predicates over R. These predicates are:

P1: Eno < 10

P2: Eno > 20

P3: Sal $\geq$ 50K

P4: Eno ≥ 20

P5: $30 <$ Eno < 60

P6: Degree = Ph.D.

P7: Eno < 15

P8: Eno > 50

P9: Sal < 50K

Their corresponding transactions are shown as follows:

T1: Eno < 10, Sal ≥ 50K, Degree = Ph.D.

T2: Eno < 20, Sal ≥ 50K

T3: Eno ≥ 20, Sal ≥ 50K

T4: $30 <$ Eno < 60, Sal < 50K, Degree = Ph.D.

T5: Eno < 15, Sal < 50K, Degree = Ph.D.

T6: Eno > 50, Sal < 50K, Degree = Ph.D.

T7: Eno < 15, Sal ≥ 50K

Based on the information provided, the transaction-predicate matrix shown in Figure 2.14 is produced. Since transaction 1 uses predicates P1, P3, and P6, the corresponding matrix elements are "1".

	Attributes									
	P1	P2	P3	P4	P5	P6	P7	P8	P9	
Transactions										Access Frequencies
T1	1	0	1	0	0	1	0	0	0	25
T2	0	1	1	0	0	0	0	0	0	50
T3	0	0	1	1	0	0	0	0	0	25
T4	0	0	0	0	1	1	0	0	1	35
T5	0	0	0	0	0	1	1	0	1	25
T6	0	0	0	0	0	1	0	1	1	25
T7	0	0	1	0	0	0	1	0	0	25

Figure 2.14: Initial transaction-predicate matrix

A Genetic Algorithm for Horizontal Partitioning

We apply our algorithm and obtain the following path for predicates:

3-2-1-4-8-7-5-9-6 Total cost: 485

and for the transaction group:

1-2-3-7-5-6-4 Total cost: 410

	Attributes									
	P3	P2	P1	P4	P8	P7	P5	P9	P6	
Transactions										Access Frequencies
T1	1		1						1	25
T2	1	1								50
T3	1			1						25
T7	1					1				25
T5						1		1	1	25
T6					1			1	1	25
T4							1	1	1	35

Figure 2.15: Re-arranged transaction-predicate matrix

The re-arranged transaction-predicate matrix is shown in Figure 2.15. For simplicity, only the entries with "1" are shown. Clearly, it is difficult to identify sub-matrices. However, the edge that contributes the highest cost of the predicate path is 7-5. We cut the path at this edge. As a result, we have higher cost between fragments, and within a fragment, we have lower cost among predicates. Two subsets are formed which are {P3, P2, P1, P4, P8, P7} and {P5, P9, P6}. Note that we obtain the same result as Zhang and Orlowska [114] when we apply the quadratic equation (proposed by Navathe et al. [87]) to cluster the matrix.

As proposed by Zhang and Orlowska [114], the first subset {P3, P2, P1, P4, P8, P7} can be simplified to Sal $\geq$ 50K. The second subset for fragmentation is {P5, P9,

P6}. That is (30 < Eno < 60) AND (Sal < 50K) AND (Degree=Ph.D). Thus the final predicates are:

Predicate 1: Sal $\geq$ 50K

Predicate 2: (Degree=Ph.D.) AND (Sal < 50K) AND (30 < Eno 60)

Tuples satisfying either one of the two predicates will be assigned to the corresponding partitions and those which satisfy neither will be assigned to the third partition. In this way, the third partition could be characterized by the following predicate: ((Sal < 50K) AND (Degree $\neq$ Ph.D.)) OR ((Sal < 50K) AND (Eno $\leq$ 30 OR Eno $\geq$ 60)).

Conclusions

In this section, we examine our new GA clustering algorithm for solving the horizontal partitioning (HP) problem. By using the transaction-predicate matrix, we define the cost measure between predicates. Then, we employ our new GA clustering algorithm to cluster predicates. We adopt an example from a literature to demonstrate the practicality of our algorithm in solving this problem.

2.3.3 Object-Oriented Database Design

Recently, the object-oriented (OO) data model has evolved as an alternative to the relational data model for supporting modern database applications such as office automation systems and computer aided design. As a consequence, the performance demand on OO database systems has increased. The literature has shown that high performance OODB systems can be achieved by using various data partitioning techniques. These include vertical, horizontal, mixed and path partitioning. In this thesis, we focus on vertical partitioning an OODB.

Object-oriented databases (OODBs) present additional semantics like structural properties (inheritance, composite objects) and interrelationships between objects.

Hence, the existing clustering algorithms (used in relational databases, for instance) have to be adapted to the object-oriented model. Also, the VP problem in relational database is known to be NP-hard [75]. Since OODBs present additional semantics and methods in the classes, vertical partitioning an OODB is more complex [29]. In this chapter, we demonstrate that our new genetic clustering algorithm (GA) can be adapted to solve this partitioning problem.

Related Work for partitioning an OODB

In relational databases, many vertical partitioning algorithms are based upon logical factors such as attribute-attribute affinity (**AAA**) [87] and physical factors such as minimization of disk accesses [18]. Bellatreche *et al.* [7] and Ezeife *et al.* [29] propose two algorithms respectively for vertical partitioning an OODB. Both attempt at maximizing the logical affinity between the methods (method-method affinity, **MMA**). MMA is defined between any two methods as the sum of frequencies of all the transactions which access the two methods together. Similarly, AAA is defined between any two attributes as the sum of access frequencies of all the transactions accessing the two attributes together. As pointed out by Chinchwadkar *et al.* [15], more affinity between the methods in a fragment implies reduction in the remote methods invocations and reduction in communication cost. Also, more affinity between the attributes in a fragment implies less irrelevant IO.

Ezeife's algorithm [29] tries to maximize the objective function by interchanging the columns of the MMA matrix. MMA matrix is a square matrix in which the numbers of rows and columns are the same as the number of methods in the class. Each entry in the MMA matrix represents the MMA value between the pair of methods represented by the corresponding row and column. Similarly, AAA matrix can be formed in the same manner. The objective function is a measure of affinities between columns and the two columns adjacent to them. Bellatreche's algorithm [7] identifies partitions by forming

a MMA graph and attempts to find maximum affinity cycles inside the graph. Each cycle represents a fragment. Both of these algorithms form groups of methods such that the methods which are frequently accessed together belong to the same fragment.

Since vertical partitioning problem is NP-hard [75], we explore the use of genetic algorithm (GA) for solving this problem. GA is a well-known algorithm for solving difficult combinatorial optimization problems (see Section 1.3).

The Model

Similarly, GA can be adapted to vertical partitioning of an OODB by minimizing the cost between methods (method-method cost) in each partition. We define the **MMC** (method-method cost) between any two methods as the sum of the access frequencies of all the transactions which access either of the two methods but not both. Also, **AAC** (attribute-attribute cost) between any two attributes within the same class is defined as the sum of the access frequencies of all the transactions which access either of the two attributes but not both. Note that all the access frequencies to the attributes within a class included methods that access the class directly and through subclasses or other classes permitted by the schema [29]. Also, if the sizes of attributes are known, Equation 2.2 should be modified to incorporate the weighting of attributes in the cost function. We modify Equation 2.2 to measure the distance (cost) between attribute A_i and A_j within the same class as follows:

$$d_{ij} = \sum_{k=1}^{m} |(a_{ki} \times w_i - a_{kj} \times w_j) \times F_k| \tag{2.6}$$

where a_{ki} is the entry in the unorganized transaction-attribute matrix (k represents transaction T_k and i represents attribute A_i) and F_k is the access frequencies for transaction T_k per unit time period (e.g., a day) and w_i denotes the size of the attribute A_i. Note that if transaction T_k 'uses' the attribute A_i, a_{ki} is 1, else, a_{ki} is 0. In Equation 2.6,

we also consider the sizes of attributes since large-sized attributes always consume more resources and use more IO time and communication time than small-sized attributes. Therefore, weighting which is proportional to the size of the attribute is incorporated into the cost function.

An Example

An example is adopted from literature [15] to demonstrate our algorithm in solving the VP of an OODB. Since our aim is data distribution rather than optimization of communication, we use the AAC matrix as an input. Figure 2.16 shows an example of a DEPT_EMPLOYEE database schema.

```
Class DEPT {
        string          Dname;
        EMPLOYEE        Dhead;
        int             Dno;
}
Class DATE {
        int             Day;
        int             Month;
        int             Year;
}
Class PERSON_IN_ORGN {
        string          SocSecNo;
}
Class EMPLOYEE: public PERSONS_IN_ORGN {
        string          Ename;
        string          Ecode;
        DATE            DateBirth;
        char            Sex;
        int             Salary;
        string          Designation;
}
Class M_EMPLOYEE: public EMPLOYEE {
        int             NoOfChildren;
}
```

Figure 2.16: Dept_Employee database

Suppose class EMPLOYEE is to be partitioned. The attribute sizes for this class is shown in Table 2.1. Table 2.2 shows the transactions that access attributes of this class and their corresponding access frequencies. It includes all the accesses to the attributes of class EMPLOYEE directly and through subclassses and other classes permitted by the schema [29]. Therefore, attribute accesses of some transactions may be identical (for example, T4, T7 and T15).

	Attribute	Size in bytes
E1	Ename	30
E2	Ecode	8
E3	DateBirth	8
E4	Sex	1
E5	Salary	4
E6	Designation	5

Table 2.1: Attribute sizes for class EMPLOYEE

Transactions	Attribute Accessed	Access Frequency
T1	E1 E4 E6	25
T2	E2 E5	15
T3	E1 E5	15
T4	E1 E3	5
T5	E1 E2	15
T6	E1 E2 E3	10
T7	E1 E3	10
T8	E1 E3 E5	5
T9	E1 E4	15
T10	E1 E2 E5 E6	40
T11	E4 E5 E6	15
T12	E2 E5 E6	15
T13	E5 E6	15
T14	E2 E5	15
T15	E1 E3	20

Table 2.2: Transactions that access the attributes of class EMPLOYEE

Using Equation 2.6, the cost between any two attributes of the class EMPLOYEE is computed and shown in Figure 2.17. It is an AAC matrix. An AAC matrix is a square matrix in which numbers of rows and columns are the same as the number of methods in the class. Each entry in the AAC matrix represents the AAC value between the pair of attributes represented by corresponding row and column. Once we have a cost matrix, we can apply our GA clustering algorithm to permute the rows and columns in the matrix so as to cluster similar attributes together. Note that the matrix shown in Figure 2.17 is a symmetric matrix, we only have to permute either the rows or the columns. As pointed out by Chinchwadkar *et al.* [15], more affinity between the methods in a fragment implies reduction in the remote methods invocations and reduction in communication cost. Also, more affinity between the attributes in a fragment implies less irrelevant IO. Note that our aim is to minimize irrelevant IO. If optimization of communication is needed, we have to use a MMC matrix as an input.

	E1	E2	E3	E4	E5	E6
E1	0	4640	4400	4775	4860	4500
E2	4640	0	1120	935	740	1650
E3	4400	1120	0	455	900	2050
E4	4775	935	455	0	565	1625
E5	4860	740	900	565	0	1510
E6	4500	1650	2050	1625	1510	0

Figure 2.17: AAC matrix for the class EMPLOYEE

By applying the GA with population size 1200, bias 1.2 and ECER operator (see Chapter 4) is used, we ran our program on a Sun SPARC Ultra-5_10 workstation. The computational time is less than 1 second. The re-arranged ACC is obtained and is shown in Figure 2.18. All solutions are converged and we obtain an attribute path which is {E1, E3, E4, E5, E2, E6}. The total distance is 7810. Using Stanfel's approach [102] to cluster the solution path, we identify two fragments which are {E1, E3}, {E4,

E5, E2, E6}. When the second fragment are further partitioned, we have two more fragments which are {E4, E5} and {E2, E6}.

	E1	E3	E4	E5	E2	E6
E1	0	4400	4775	4860	4640	4500
E3	4400	0	455	900	1120	2050
E4	4775	455	0	565	935	1625
E5	4860	900	565	0	740	1510
E2	4640	1120	935	740	0	1650
E6	4500	2050	1625	1510	1650	0

Figure 2.18: Re-arranged AAC matrix for the class EMPLOYEE

Conclusions

Recently, the object-oriented (OO) data model has evolved as an alternative to the relational data model for supporting modern database applications such as office automation systems and computer aided design. As a consequence, the performance demand on OO database systems has increased. However, OODB presents additional semantics like structural properties (inheritance, composite objects) and interrelationships between objects, the existing clustering algorithms have to be adapted to the OO model. In this section, we examine our new GA clustering algorithm for vertical partitioning an object-oriented database (OODB). We introduce the method-method cost (MMC) between two methods in a class and attribute-attribute cost (AAC) between any two attributes within the same class. The MMC is used to optimize communication while the AAC is used to minimize irrelevant IO. In this thesis, we focus on the AAC (i.e. data distribution) and adopt an example from a literature to demonstrate the practicality of our algorithm in solving this problem.

2.3.4 Document Database Design

Due to the rapid growth of the World Wide Web and hardware performance, many database systems are built for storing documents. These databases, named document databases, increase the convenience for query and access. A document databases usually consists of a large number of electronic books and a major portion in the digital library is the electronic books. Also, there has been a recent trend to publish electronic books rather than hard copy. Especially in the professional field, the reference manuals are usually preserved as the document databases in order to increase the convenience to query. Therefore, research relating to digital library has become an important issue.

If there is a document database system, which is modeled as shown in Figure 2.19, users can formulate a query to retrieve documents, and re-formulate the query when they see the results, and so on, until satisfied with the answer [86]. The query effectiveness depends upon user's knowledge about the query language. In order to improve the efficiency of a document database, similar documents should be clustered together and stored in the same site(s)/machine(s) in a computer network. Obviously, the above problem is a partitioning problem which can be tackled by our new genetic clustering algorithm (GA).

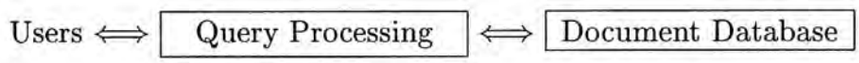

Figure 2.19: The model of a document database system

Structural Document Databases

Classical information retrieval on document databases usually allows little structuring [86], since it retrieves information only on data. In order to improve the performance of the document databases, documents that stored in the database should be structurally organized. In addition to the electronic form of content that stored in the

database, structural information such as chapter, section and paragraph hierarchy may also embedded in the database. Such structural information are useful in querying the documents which named the *Structural Documents* (SD) [59], since most people always read books with chapter-oriented concept.

Jiang *et al.* [58] propose an idea to transform the documents into a set of structural documents, which merge two documents with similarity greater than the given threshold into one structural document. Based on this idea, they develop a clustering-based approach to construct the SD. Similarly, we define the cost measure between any two different documents and develop a GA-based clustering approach to cluster documents. If similar documents based on its structure are partitioned together, the retrieval and query time of the database can be improved.

The Model

To measure the cost between two documents, we use the heuristics similar to the similarity measure developed by Jiang *et al.* [58]. The cost measure between two documents in two different chapters is higher than that in the same chapter, and the cost between two different documents in two different sections is higher than in the same section. Without loss of generality, we follow the assumptions made by Jiang *et al.* [58], which assume the whole reference book to be divided into a three-tier hierarchy, including chapter, section and paragraph. Based on these heuristics, we define the *Hierarchy Independence* (HI) between two documents, which can be easily computed by the following procedure:

Step 1: If two documents are in the same chapter, HI $\leftarrow$ 0 and stop.

Step 2: If two documents are in the same section, HI $\leftarrow$ (1/s), where s is the total number of sections in this chapter.

Step 3: If two documents are in the same paragraph, HI $\leftarrow$ 0.5 + (1/p), where p is the total number of paragraphs in this section, and stop.

Step 4. HI $\leftarrow$ 1.

Figure 2.20: Procedure for computing HI between two documents

Let the two documents be denoted as D_i and D_j. The cost of D_i and D_j, denoted by $C(i,j)$, is computed by the following formula:

$$C(i,j) = (1-\delta) \times \text{different}(i,j) + \delta \times HI(i,j), \tag{2.7}$$

where different(i,j) means the number of words and keywords which appear in either of documents D_i and D_j, but not both. The value is normalized by dividing the total number of keywords in both documents D_i and D_j. $HI(i,j)$ is the hierarchy independence of documents D_i and D_j and δ is an adaptive weight value with $0 \leq \delta \leq 1$.

The default value of δ is 0.5, which can be adjusted by the number of chapters for a given book. For example, when the number of chapters is near to 1 or n for a book divided into n documents, δ is set as the value closed to 0, as there is not much meaning in the structure [58].

An Example of Cost Measure

An example is adopted from Jiang *et al.* [58] to demonstrate the cost measure between two different documents. Let D_i = {'intelligent', 'query', 'agent'}, $|D_i| = 3$, and D_j = {'database', 'query'}, $|D_j| = 2$. Therefore, the number of the words which appear either in documents D_i and D_j but not both is 3, and we have different$(i,j) = 0.6$.

By computing the cost measure between any two different documents D_i and D_j,

a cost matrix $[C_{ij}]$ can be formed by letting $C_{ij} = C(i, j)$. GA can be applied to the matrix $[C_{ij}]$ and minimize the cost of the sequence of C_{ij} by re-arrange the rows and columns of the matrix. After we obtain the re-arranged matrix, we can identify groups of documents by using the Stanfel's clustering method [102].

A Clustering Example

Assume we have six documents, {D1, D2, D3, D4, D5, D6}, and the cost matrix is shown in Figure 2.21.

	D_1	D_2	D_3	D_4	D_5	D_6
D_1	0	0.2	0.6	0.5	0.6	0.9
D_2	0.2	0	0.5	0.4	0.7	0.8
D_3	0.6	0.5	0	0.3	0.2	0.9
D_4	0.5	0.4	0.3	0	0.1	0.7
D_5	0.6	0.7	0.2	0.1	0	0.8
D_6	0.9	0.8	0.9	0.7	0.8	0

Figure 2.21: Cost matrix with six documents

By applying GA with population size 1200, bias 1.2 and ECER operator (see Chapter 4) is used, we obtain the re-arranged matrix and is shown in Figure 2.22. The solution sequence {D1, D2, D3, D5, D4, D6} which has total cost of 1.7. We further clustered the sequence using the Stanfel's clustering approach [102]. Three fragments are identified which are {D1, D2}, {D3, D5} and {D4, D6}.

	D_1	D_2	D_3	D_5	D_4	D_6
D_1	0	0.2	0.6	0.6	0.5	0.9
D_2	0.2	0	0.5	0.7	0.4	0.8
D_3	0.6	0.5	0	0.2	0.3	0.9
D_5	0.6	0.7	0.2	0	0.1	0.8
D_4	0.5	0.4	0.3	0.1	0	0.7
D_6	0.9	0.8	0.9	0.8	0.7	0

Figure 2.22: Re-arranged cost matrix with six documents

Remarks on Chinese document database: Besides English document database, GA is capable of partitioning Chinese document databases by using the association measure developed by (Liang [73]; Sproat and Shih [101]). Since there is no obvious word boundary in the Chinese text, an identification process for identifying each possible disyllabic word from target database is needed. After the disyllabic words are identified from the sentence by applying the association measure, each document can be transferred to a set of keywords. After this transformation, cost measure can be proceeded as usual [58].

Conclusions

Due to the rapid growth of the World Wide Web and hardware performance, many database systems are built for storing documents. These databases, named document databases, increase the convenience for query and access. Classical information retrieval on document databases usually allows little structuring. However, structural information is useful in designing a document database, for instance, most people always read books with chapter-oriented concept. Besides, the literature has shown that documents that stored in the database can be structurally organized. Based on this idea, we introduce the cost measure for documents and employ our new GA clustering algorithm to partition the document databases. We adopt an example from a literature to demonstrate the practicality of our algorithm in solving this problem.

2.4 Conclusions

In this chapter, we review the TSP literature and explore the use of genetic algorithms (GA) to solve several clustering problems in information systems. They are vertical partitioning (VP), horizontal partitioning (HP), object-oriented database (OODB) design, and document database design. For each application, we review the related literature and formulate a solution model for it. An example is also used to demonstrate the

practicality of the proposed algorithm to the underlying application.

In Section 2.3.1, we present the vertical partitioning (VP) problem. We then formulate the problem as a traveling salesman problem (TSP) and propose a methodology based on genetic algorithms (GA) to solve the corresponding TSP-VP. We also adopt an example from a literature to demonstrate the practicality of our new GA clustering algorithm in solving the VP problem.

In Section 2.3.2, we examine our new GA clustering algorithm for solving the horizontal partitioning (HP) problem. By using the transaction-predicate matrix, we define the cost measure between predicates. Then, we employ our new GA clustering algorithm to cluster predicates. We adopt an example from a literature to demonstrate the practicality of our algorithm in solving this problem.

In Section 2.3.3, we examine our new GA clustering algorithm for vertical partitioning an object-oriented database (OODB). We introduce the method-method cost (MMC) between two methods in a class and attribute-attribute cost (AAC) between any two attributes within the same class. Based on these cost functions, our proposed algorithm is adapted to cluster fragments. We adopt an example from a literature to demonstrate the practicality of our algorithm in solving this problem.

In Section 2.3.4, we introduce the cost measure for documents and employ our new GA clustering algorithm to partition the document databases. We adopt an example from a literature to demonstrate the practicality of our algorithm in solving this problem.

Chapter 3

The Experiments for Vertical Partitioning Problem

3.1 Introduction

To evaluate the performance of our proposed algorithm, we compare our approach with the Slagle's [100], an efficient heuristic method for solving the Vertical Partitioning (VP) problem. It should be noted that the Slagle's algorithm is similar to the bond energy algorithm (BE) [80] which is a well-known clustering algorithm. Most of the current approaches ([12], [18], [87], [114]) for database partitioning problem apply BE as a clustering method.

In the next section, we first demonstrate the disadvantages of the BE and show that the same problems would not occur when our proposed algorithm is used. Next, in Section 3.3, we generate several VP problems whose sizes range from 20 to 100 attributes. By solving these problems, we compare the performance of our proposed algorithm to the Slagle's.

3.2 Comparative Study

The Bond Energy Algorithm (BE) is proposed by McCormick *et al.* [80] in 1972. The algorithm shown in Figure 3.1 is a straightforward algorithm for permuting the rows and columns of an $M \times N$ matrix A of nonnegative entries so as to maximize the objective function.

Step 1: Place one of the rows of an $m \times n$ array arbitrarily. Set $i = 1$.
Step 2: Set $j = i + 1$.
Step 3: Place the jth row in each of the $i + 1$ positions, and compute the row's contribution to the measure of effectiveness[1].
Step 4: $j = j + 1$ and repeat Step 3 until $j = m$.
Step 5: Place the row j at the position that gives the largest incremental contribution to the measure of effectiveness.
Step 6: $i = i + 1$ and repeat Steps 2-6 until $i = m$.
Repeat the above steps for the columns.

Figure 3.1: The bond energy algorithm

Later, Slagle *et al.* [100] modify the BE and use it in the clustering problem. The algorithm of Slagle *et al.* is quite similar to the BE. The difference is that the objective function is maximized in BE, while in Slagle's algorithm, the objective function is minimized. In BE, it measures the affinity between every objects, whereas in Slagle's algorithm, it measures the cost between every objects. The algorithm of Slagle *et al.* in Figure 3.2 can find a short path, but not necessarily the shortest.

[1]The measure of effectiveness of an array A is the sum of the bond strengths in the array, where the bond strength between the two nearest neighbor elements is defined as their product.

Step 1: Place one of the rows of an $m \times n$ array arbitrarily. Set $i = 1$.
Step 2: Arbitrarily select a row from the remaining $m - i$ rows.
Step 3: Place the row in each of the $i + 1$ positions and compute the row's contribution to the weight of the path.
Step 4: Place the row at the position that gives the smallest incremental contribution to the weight of the path [100].
Step 5: $i = i + 1$ and repeat Steps 2-5 until $i = m$.
The above steps are repeated for the columns as well.

Figure 3.2: The algorithm of Slagle *et al.*

In the work of Navathe *et al* [87], the Bond Energy algorithm [80] is first applied to permute rows and columns of a transaction-attribute matrix, then a hierarchical clustering algorithm called SPLIT_NON_OVERLAP is applied to further partition the matrix into two nonoverlapping fragments. Their hierarchical clustering algorithm is simply to maximize a quadratic function so that fragments produced are "balanced" with respect to transaction load. However, the SPLIT_NON_OVERLAP algorithm has the disadvantage of not being able to partition an object by selecting out an embedded "inner" block. An example is given in Figure 3.3. Clearly, {A2, A3} should form a fragment and {A1, A4, A5} should form an another fragment. Obviously, they cannot be identified by using the SPLIT_NON_OVERLAP algorithm and existing hierarchical algorithms cannot handle them effectively.

	Attributes					
	A1	A2	A3	A4	A5	Access Frequencies
Transactions						
T1	1			1	1	20
T2		1	1			20
T3		1	1			20
T4	1				1	20

Figure 3.3: Transaction-attribute matrix 4 with an embedded "inner" block

The disadvantage of the Bond Energy algorithm is that it may produce transaction-attribute matrix with "inner" block structure. It is because the Bond Energy algorithm used in permuting transactions/attributes are unable to select the most contributing transaction/attribute to be the starting and ending transaction/attribute in the sequence. Hence, the solution sequence obtained may need to be further adjusted.

To correct this problem, they propose a procedure called SHIFT which moves the leftmost column of the matrix in Figure 3.3 to the extreme right, and the topmost row of the matrix to the bottom (see Figure 3.4) so that every diagonal block gets the opportunity of being brought to the upper left corner in the matrix.

	Attributes					
	A2	A3	A4	A5	A1	Access Frequencies
Transactions						
T2	1	1				20
T3	1	1				20
T4				1	1	20
T1			1	1	1	20

Figure 3.4: Transaction-attribute matrix 5 after the SHIFT procedure is applied

Same problem would not occur when GA is used to permute rows and columns in a transaction-attribute matrix for GA has already "considered" the best way to select the most contributing transaction/attribute to be the starting transaction/attribute. In this way, the whole sequence is benefited (the total distance of the sequence is minimized). Recall that a dummy transaction/attribute is introduced into the TSP so that the solution tour obtained is already the shortest sequence (path).

3.3 Experimental Results

To compare the effectiveness of our algorithm with the Slagle's, several problem instances with different sizes are generated. First, we generate a matrix with diagonal block structure. Next, each transaction is randomly assigned an access frequency. To avoid solving a trivial problem, some randomness is introduced into the matrix. By taking some chances, say 0.1, the entries in the matrix are swapped from 0 to 1 or from 1 to 0. Finally, the columns and rows of the matrix are randomly swapped and an initial transaction-attribute matrix is generated.

Each problem is solved ten times and the average is obtained. For GA, the population size is set to 1200 and selective pressure to 1.2 and ECER operator is used (see Chapter 4). The program is run on a SUN SPARC Ultra-5_10 workstation. Results for solving VP are shown in Table 3.1. Note that our aim is to minimize the total distance of TSP and the summation values of Equation 2.5.

Problem Size	Number of fragments	Probability of swapping	Total distance of the solution by GA	Total distance of the solution by Slagle	Values of Eq. 2.5 by GA	Values of Eq. 2.5 by Slagle
20	4	0	573	597.6	-481	-481
20	4	0.1	1817	3336.6	-455	-380.9
20	4	0.2	2343	4479.9	-519	-361.1
20	5	0.1	1190	2411.9	-420.6	-379.9
20	5	0.2	1542	2924.2	-449	-388.5
50	5	0.1	10099.8	22785.3	-1200.2	-712
50	5	0.2	15171.2	28332.7	-711.5	-536.7
50	10	0.1	15906.5	29391.7	-2017.2	-1349.1
50	10	0.2	26551.9	38786.2	-2113.9	-1752.2
100	10	0.1	31796.4	61023.1	-1662.5	-974.2
100	10	0.2	49335.1	81008.1	-1564.8	-1155.3
100	20	0.1	20223.1	38319.5	-2583	-1886.3
100	20	0.2	33403.1	58609.8	-3457	-2564.5

Total distance of the solution: the summation of the values of Equation 2.2
Value of Equation 2.5: the summation of the objective values of Equation 2.5

Table 3.1: Results for solving VP with GA and Slagle's algorithm

In Table 3.1, the problem size is the number of attributes of the VP. Number of fragments is the number of clusters that the problem has. The third column shows the probability of swapping. It is used to control the randomness of the problem. Large swapping value suggests that the transaction-attribute matrix is very scattered. Note that for swapping value larger than 0.2, the problem become very scattered and very few interesting clusters can be formed. Therefore, the swapping value is restricted to less than 0.2.

The total distance of the solution path is the summation of the values as described in Equation 2.2. Note that the distance used by GA is the same as by the Slagle's. When clusters are identified using Stanfel's approach, the objective values of Equation 2.5 are also summed up. Small values of Equation 2.5 suggest that clusters obtained are more valid (see Section 2.3.1). It is because the value of Equation 2.5 measures the difference between the average distance within fragments and the average distance between fragments.

The first row in Table 3.1 shows a problem with 20 attributes and 4 fragments. The swapping value is zero. That means, the matrix can be re-arranged to a perfectly diagonal block structure and each transaction will access one fragment only. The summation values of Equation 2.5 are both -481, which indicate that both algorithms are able to cluster similar attributes together and identify the corresponding fragments. Regarding the objective value, GA has a lower value, which indicates that GA can further minimize the total distance of the attribute path.

For other problems, GA further minimizes the objective function and gives lower values of Equation 2.5. This implies that fragments clustered by GA is more reasonable. Compare with other methods, GA is more significant in solving large and scattered problems.

3.4 Conclusions

In Section 3.2, we demonstrate the disadvantage of the Bond Energy algorithm (BE) [80]. BE may produce transaction-attribute matrix with "inner" block structure. Note that there is no efficient hierarchical clustering algorithm which can partition an embedded "inner" block matrix. In the work of Navathe *et al.* [87], an additional procedure known as SHIFT is proposed as the remedy. It is invoked before a hierarchical clustering algorithm can be applied. However, we have shown that this problem will not occur when the proposed algorithm is used to permute transaction/attribute in a transaction-attribute matrix. After the proposed algorithm is applied, any existing splitting algorithms can be used to identify fragments.

In Section 3.3, we compare the Slagle's algorithm [100], which is a modified version of the Bond Energy algorithm [80], with the proposed algorithm. Experimental results indicate that, with the same objective function, our proposed algorithm outperforms the Slagle's in the value of the objective function obtained. Also, a statistics, which is similar to the Hubert's Γ statistics [53], is used to test the validity of the clusters obtained by the two algorithms. Computational results shows that the clusters obtained by our proposed algorithm is more reasonable than the Slagle's. The performance of our proposed algorithm is more significant in solving large-scale problems. Unlike Eisner and Serverance [28] and some heuristic algorithms, GA can solve large scale problem very well as it always guarantees a fair solution under finite computation time. It is critical in real life situations, e.g. banking, where the number of transaction could be thousands and the number of attributes could be up to hundreds.

Chapter 4

Three New Operators for TSP

4.1 Introduction

As stated by Falkenauer [30], perhaps the most important technique in Genetic Algorithms (GA) is the crossover operator, also called the recombination operator. Thus, an effective and reliable crossover operator can greatly enhance the system. In this chapter, we develop three new crossover operators which can satisfy these criteria.

Whitley *et al.* [112] devise a crossover operator called Edge Recombination (ER) for solving the traveling salesman problem (TSP), which is based on Grefenstette's [41] but has a better edge-preserving property. Later, Starkweather *et al.* [103] develop an improved version of ER (EER) which can further improve the system. Their work concludes that the effectiveness of different operators is dependent on the problem domain: operators which work well in problems where adjacency is important (e.g. TSP) may not be effective for other types of sequencing problems. Also, operators which perform poorly on the TSP work extremely well for the warehouse scheduling task. ER and EER emphasize edges in that 95% to 99% of the edges that compose the offspring are inherited from one of the two parents. So the population can converge at a faster rate without losing important adjacency information (good schemata). In case

when edge failure occurs (i.e. isolated city occurs), the ER crossover can randomly choose a new city to continue the tour. This provides an effective mutation rate of 0.009, or less than 1% [112]. For enhanced ER, the average trials needed are even fewer which suggests that EER can transfer edges from parents more effectively [103]. Thus, ER and EER outperform its predecessors for solving TSP.

However, we believe that the edges transformation process can be done better if we have more information on the choice of edges that are needed to be transferred. In this chapter, we devise three new operators for TSP which are capable of effectively using adjacency information as EER and at the same time improve the inheritance of the mating process.

4.2 Enhanced Cost Edge Recombination Operator

We propose a modified version of EER in this section. Our proposed crossover (ECER) is similar to the Enhanced Edge Recombination crossover operator (EER) [103] except that we consider the cost of edges for breaking ties. It has been observed that GA works well on TSP if adjacency information from two parents can be effectively transferred to offspring. If tie breaking in EER is managed well, the process of transformation of adjacency information from parents can be done more effectively.

The Edge Recombination (ER) operator is different from other genetic sequencing operators in that it emphasizes adjacency information instead of the order or position of the items in the sequence. The "edge table" used by the operator is really an adjacency table listing the connections in and out of a city found in the two parent sequences. The edges are then used to construct offspring in such a way that isolation cities or elements are avoided in the sequence [103].

For example, the tour [b a d f g e c j h i] contains the links [ba, ad, df, fg, ge, ec, cj, jh, hi, ib]. In order to preserve links present in the two parent sequences, a table is built which contains all the links present in each parent tour. Building the offspring

then proceeds [112] as described in Figure 4.1.

1. Select a starting element. This can be one of the starting elements from a parent, or can be chosen from the set of elements that have the fewest entries in the edge table.
2. Of the elements that have links to this previous element, choose the element that has the fewest number of links remaining in its edge table entry. Ties are broken randomly.
3. Repeat step 2 until the new offspring sequence is complete.

Figure 4.1: The algorithm for Edge Recombination (ER) operator

Consider the following sequences: [a b c d e f] and [c d e b f a]. An edge table of Edge Recombination is given in Table 4.1.

city	*link*			
a	b	f	c	
b	a	c	e	f
c	b	d	a	
d	c	e		
e	d	f	b	
f	e	a	b	

Table 4.1: Edge table for Edge Recombination (ER) operator

Suppose element a is selected to start the offspring tour. Since a has been used, all occurrences of a are removed from the right-hand side of the edge table. Element a has links to elements b, f and c. Element f and c both have 2 links remaining in their table entries. Therefore, f is randomly selected as the next element in the offspring, and all occurrences of f are removed from the right-hand side of the edge table. Element f has links to e and b, both of which have 2 links remaining. Therefore, either one of them is selected randomly and the process continues until the child tour is complete.

The offspring is then produced as [a f b c d e].

When the Edge Recombination operator was first implemented, it had no active mechanism to preserve "common subsequences" between two parents. Later, Starkweather *et al.* [103] propose the Enhanced Edge Recombination (EER) operator which solve this problem. During the construction of the "edge table" in EER, if an insertion involves an item, which is already in the edge table, that element of the sequence must be a common edge. The elements of a sequence are stored in the edge table as integers; so if an element is already present, the sign of the value is inverted to represent a common edge: e.g. if A is already in the table, change it to $-A$. The sign acts as a flag. Consider the sequences [a b c d e f] and [c d e b f a] and edge table in Table 4.2.

city	*link*			
a	b	-f	c	
b	a	c	e	f
c	b	-d	a	
d	-c	-e		
e	-d	f	b	
f	e	-a	b	

Table 4.2: Edge table for Enhanced Edge Recombination (EER) operator

In EER, priority is given to negative entries when constructing offspring. Suppose the starting city is city a, EER will choose city f as its next city. However, a tie can be found when there are more than one entry with negative values and both having the same number of links to other entries. In this circumstance, EER will break the tie randomly. But for ECER, it will choose the edge with the smallest cost among those entries. Suppose the starting city is city d. Refer to the edge table, there are two entries (c and e) with negative values. Besides, both of them have three links to other entries. EER would break the tie randomly by choosing either city c or city e as its adjacent city. However, ECER will choose the edge with minimum cost. Suppose the

edge [dc] having edge cost of 100 and edge [de] having cost of 40. In this case, city e will be chosen as its adjacent city.

Also, if there is no negative entries, ECER will choose the entry with the smallest number of links to other entries. If a tie is found, it will break the tie by choosing the edge with the smallest cost. The algorithm for ECER is given in Figure 4.2.

1. Select a starting element. This can be one of the starting elements from a parent.
2. Construct the edge table. Assign negative values to common edges.
3. Of the elements that have links to this previous element, choose the element that has the fewest number of links remaining in its edge table entry, priority is given to negative entries.
4. If a tie is found, break it by choosing the edge with the smallest cost. When there are more than one edge with the smallest cost, break the tie randomly.
5. Repeat step 3-4 until the new offspring sequence in complete.
6. Select the first element from the other parent as a starting element, repeat the above steps to generate the second offspring.

Figure 4.2: The algorithm for Enhanced Cost Edge Recombination (ECER) operator

Note that ECER will bias the crossover permutation by favouring the selection of edges with lower costs. It speeds up the searching time for the whole system without degrading the quality of the solution. Some experiments are conducted to demonstrate that ECER outperforms the EER in most cases (see Section 4.5).

4.3 Shortest Path Operator

In this section, a new genetic sequencing operator, namely Shortest Path crossover operator (SP), is proposed. As stated by Holland (1975) [50], during the reproductive phase of GA, individuals are selected from the population and recombined, producing

offspring which will comprise the next generation. Therefore, good individuals will probably be selected several times in a generation, poor ones may not be at all. However, it has been observed that crossover may produce offspring of low fitness. Although these offspring will not be likely to get selected for reproduction in next generation, the overall fitness of the population is decreased. Since the purpose of crossover is to produce better offspring, this effect may decrease the efficiency of the system. Hence, to fully optimize the mating process, we develop the SP operator which seek to generate a local optimal offspring in each generation.

The crossover permutation can be considered as a shortest path problem. The pervious example (Table 4.2) is used to demonstrate the mechanism of SP. Suppose in a particular generation, two parents [a b c d e f] and [c d e b f a] are chosen. The weight matrix is constructed in Figure 4.3.

	a	b	c	d	e	f
a	—	80	50	∞	∞	40
b	80	—	20	∞	20	30
c	50	20	—	100	∞	∞
d	∞	∞	100	—	40	∞
e	∞	20	∞	40	—	25
f	40	30	∞	∞	25	—

Figure 4.3: Weight matrix for SP operator

If there is no corresponding edge in both parents, the cost of the edge is set to ∞ (in practice, some very large number). Note that the diagonal entries are set to null because it is not applicable. By using the weight matrix, the Dijkstra's shortest path algorithm [25] is applied for finding a shortest path from a specified city to another specified city. First, let $cost(v, u)$ be the weight of edge [vu]. If no edge between v and u, $cost(v, u) = \infty$. Also, let $d(v)$ be the distance from the source to v. The algorithm

of Dijkstra is outlined in Figure 4.4.

Let V contain the source vertex and
U contain all the other vertices.
while (U is not empty) do
    choose "u" such that it is in U with smallest $d(u)$;
    add "u" to V and remove "u" from U;
    for each "w" in U do
        if $d(u) + cost(u, w) < d(w)$ then
            $d(w) = d(u) + cost(u, w)$
        end if
    end for
end while

Figure 4.4: The Dijkstra's shortest path algorithm

To construct new offspring, we proceed as shown in Figure 4.5.

1. Construct the weight matrix using the edge information from parents.
2. Choose a starting city that can be one of the starting elements from a parent.
3. Choose the ending city by using the same criterion used in ECER.
4. Apply the Dijkstra's shortest path algorithm to find a shortest path from the starting city to the ending city.
5. Update the weight matrix to remove any visited cities.
6. The ending city now becomes the starting city.
7. Repeat step 3-6 until the new offspring sequence is complete.
8. Select the first element from the other parent as a starting element, repeat the above steps to generate the second offspring.

Figure 4.5: The algorithm for Shortest Path (SP) operator

In some cases, a feasible ending city may not be found. This happens when the starting city is isolated. In this case, an unvisited city will be selected according to its

parent order to continue the sequence. Suppose we choose city a from parent 1 (see Figure 4.3) as the starting city (the first offspring). Since edge [af] is a common edge, city f will be chosen as the ending city. By applying the shortest path algorithm. The shortest path from city a to city f is simply [af]. Now, city f becomes the starting city, and according to the mechanism of ECER, it will choose city e as its ending city. The shortest path from city f to city e is just [fe]. By repeating this process, SP generates a new offspring [a f e d c b].

The searching process can be improved by using an edge table (see Table 4.2) instead of a weight matrix. The computational cost associated with SP crossover can be lesser than the Dijkstra's shortest path algorithm [25]. The original Dijkstra's shortest path requires $O(n^2)$ computational time because it requires to search n number of cities twice. With the use of an edge table, since each city can only have four feasible links, the searching time can be reduced to $O(n)$. In each generation, we might have to apply the shortest path algorithm several times. For the worst case, it may be n times for n number of cities. Therefore, the SP crossover operator will require $O(n^2)$ time for n number of cities in each generation.

Some experiments are conducted and the results are shown in Section 4.5. It demonstrates that in most cases, the quality of the solutions obtained by using SP outperform others.

4.4 Shortest Edge Operator

Our SP crossover has high computational requirement. Sometimes, the user may seek to have a fair solution and want to spend the minimum of time on computation. We propose the Shortest Edge crossover operator (SE) which may satisfy this criterion.

The mechanism for SE is very simple. By starting with a city, each time it will pick up an edge which has the minimum cost among all feasible edges. In case if no feasible edges exist, it will randomly pick up an unvisited city and continue the search. The

algorithm is given in Figure 4.6.

1. Select a starting element. This can be one of the starting elements from a parent, or can be chosen from the set of elements that have the fewest entries in the edge table.
2. Of the elements that have links to this previous element, choose the element that has the smallest distance to the previous element, breaking ties randomly.
3. Repeat step 2 until the new offspring sequence is complete.
4. Select the first element from the other parent as a starting element, repeat the above steps to generate the second offspring.

Figure 4.6: The algorithm for Shortest Edge (SE) operator

Using the same example in Section 4.3. Suppose we have two parents [a b c d e f] and [c d e b f a] and the weight matrix is constructed as in Figure 4.3. We choose city a as the starting city. Among all feasible edges [ab, ac, af], edge [af] has the minimum cost which is 40. Thus, we pick edge [af]. Starting with city f, we choose city e since edge [fe] has the minimum cost. By repeating this process, SE generates a new offspring [a f e b c d].

With the use of an edge table (Table 4.2), the searching time for the above process can be improved. Since each city can only have four feasible edges at most. Thus, for n number of cities, SE will require $O(n)$ time in each generation. Since SE will only choose the smallest edge to construct offspring, the population can be converged faster than other operators can; i.e. it gives the smallest amount of execution time. Some experiments are conducted. Results indicate that SE produces fair solution in the smallest amount of time (see Section 4.5).

4.5 The Experiments

Ten crossover operators: EER, ER, OX, PMX, CX, OX2, PB, ECER, SP and SE (see Section 2.2.2) are studied in our tests. We would like to evaluate the effectiveness of various operators in solving the traditional TSP. In particular, the performance of our operators: ECER, SP and SE should be noted.

4.5.1 Experimental Results for a 48-city TSP

A 48-city problem (hk48) is adopted from TSPLIB [95] to demonstrate the effectiveness of our proposed crossover in solving the TSP. The optimal solution for this problem is 11461 [95].

We compare the performance on two levels, each of the above operators is used to run using the same parameters for 10 experiments and then each is tuned for best results. The parameters for the first comparison are: selective pressure (bias) of 1.5 (see Section 2.2.2) and population size of 1000. We run the experiments on a SUN SPARC Ultra-5_10 machine. The results appear in Table 4.3.

Operator	Selective pressure	Population	Average trials	Average Best cost	Gap	Average time used (seconds)
SP	1.5	1000	10471.67	11493.67	0.29%	65
ECER	1.5	1000	9191.56	11515.33	0.47%	60
SE	1.5	1000	7453.78	11601.00	1.22%	24
EER	1.5	1000	20674.67	11984.22	4.37%	118
ER	1.5	1000	55240.11	12311.44	6.91%	150
OX	1.5	1000	85495.33	12493.00	8.26%	46
OX2	1.5	1000	52552.00	14113.00	23.14%	45
PB	1.5	1000	59384.67	14162.89	23.57%	62
PMX	1.5	1000	30123.56	15603.00	36.14%	31
CX	1.5	1000	15388.56	24026.56	52.30%	7

Gap: the percentage difference between our solution and the optimal solution

Table 4.3: Results for a 48-city TSP (untuned)

We attempt to optimize the performance of each operator by tuning the bias and population size. Each time we increase the population size by 100 until there is no further improvement in the solution. Next, we vary the bias from 1.1 to 2.0 and obtain the best value for the bias. The results are shown in Table 4.4.

Operator	Selective pressure	Population	Average trials	Average Best cost	Gap	Average time used (seconds)
SP	1.6	1300	11389.78	11470.89	0.09%	109
ECER	1.2	1200	12423.44	11476.33	0.13%	90
SE	1.5	1300	12867.89	11581.56	1.05%	42
EER	1.3	1400	36048.67	11604.89	1.34%	177
ER	1.1	1700	125801.40	11734.00	2.33%	381
OX	1.5	1500	145355.90	12021.33	4.66%	109
OX2	1.1	2000	175898.20	12609.22	10.02%	196
PB	1.5	1700	105145.20	13290.11	15.96%	114
PMX	1.1	1900	91963.00	14021.44	22.34%	101
CX	1.6	1300	18554.33	24083.89	52.41%	8

Gap: the percentage difference between our solution and the optimal solution

Table 4.4: Results for a 48-city TSP (tuned)

We observe that using larger population sizes lead to better solutions but more searching time is required and using higher selection bias values in general gives smaller number of trials (generations). Table 4.4 clearly indicates that Enhanced Edge Recombination operator (EER) outperforms its predecessors. CX produces the poorest results in solving the problem. In fact, the population converges too fast to a local optimum before the global optimum can be explored. In most cases, CX only uses less that 8 seconds to complete the search. For OX, OX2, PB and PMX, they produce better results after the parameters are tuned. However, their performances are still lagged behind ER and EER. Our finding is consistent with Starkweather *et al.* [103], which suggests that EER turns out to have a better performance than most order-based crossover operators

on TSP.

After the parameters are tuned (Table 4.4), ER and EER produce the solutions with gaps around 2%. As stated by Whitley *et al.* [112], these two operators emphasize edges in that 95% to 99% of the edges that compose the offspring are inherited from one of the two parents. So the population can converge at a faster rate without losing important adjacency information (good schemata). Such adjacency information is important in solving TSP [103]. In case when edge failure occurs (i.e. isolated city occurs), the ER crossover can randomly choose a new city to continue the tour. This provides an effective mutation rate of 0.009, or less than 1%. For enhanced ER, the average trials needed are even fewer which suggests that EER can transfer edges from parents more effectively [112].

Although EER is already very efficient, our proposed operators still provide visible improvement in solving this example. Using smaller production sizes and running time, our proposed operators still obtain better solutions than EER. For SE, it produces the average solution with gap of 0.13% and at the same time spent the smallest computational time. For SP, it produces the best average solution with gap of 0.09%. It is very near to the optimal solution. For ECER, the performance is in between SE and SP.

4.5.2 Experimental Results for Problems in TSPLIB

To further demonstrate the effectiveness of SE, SP and ECER, we use all the TSP instances, whose sizes are below 1000, from the TSPLIB [95]. We compare the performance of our operators against Enhanced ER. For each problem instances, we conduct the experiment once.

With the setting of parameters obtained in Table 4.4, four operators are used to solve the problem instances. To give a fair comparison, we partition the problems into 3 sets. The first set contains problem sizes less than or equal to 100 cities, set 2 contains problems with size between 100 and 500 cities, and set 3 contains problems above 500

cities. The results are shown in Table 4.5 to 4.7.

Problem name	Size of problem	Optimal solution	EER gap (%)	SE gap (%)	ECER gap (%)	SP gap (%)	EER Time (mins)	SE Time (mins)	ECER Time (mins)	SP Time (mins)
att48	48	10628	2.98	5.60	1.96	0.19	2.93	0.63	5.38	1.80
bayg29	29	1610	0.92	4.05	0.80	0	0.92	0.73	0.72	0.92
bays29	29	2020	0.69	2.98	0	0	0.95	0.77	0.88	0.50
berlin52	52	7542	2.77	4.19	0	0.07	3.75	0.62	1.40	1.62
bier127	127	118282	9.49	4.39	3.56	2.21	41.48	9.08	11.77	17.53
brazil58	58	25395	3.32	3.25	0.84	1.86	4.80	1.72	3.33	2.15
burma14	14	3323	0	0	0	0	0.18	0.17	0.10	0.15
dantzig42	42	699	2.65	7.42	0.14	0	2.27	0.45	3.82	1.02
eil51	51	426	1.16	7.19	0.23	0.23	5.08	0.70	1.42	4.55
eil76	76	538	427	3.41	0.92	2.18	10.95	1.30	5.20	19.23
fri26	26	937	0	0	0	0	0.57	0.12	0.35	0.32
gr17	17	2085	0	2.98	0	0	0.23	0.12	0.22	0.20
gr21	21	2707	0	5.55	0	0	0.35	0.10	0.23	0.38
gr24	24	1272	0	0.47	0	0	0.48	0.17	0.37	0.92
gr48	48	5046	1.69	7.0	2.62	0.90	3.08	1.05	4.57	2.10
gr96	96	55209	10.62	8.07	3.09	2.46	18.92	19.63	15.68	102.3
hk48	48	11461	5.83	0.68	0	0	3.53	0.88	1.12	1.83
kroA100	100	21282	16.54	6.70	3.81	2.41	24.83	3.15	6.10	7.73
kroB100	100	22141	16.09	6.71	4.57	5.02	34.03	2.08	9.58	63.58
kroC100	100	20749	16.97	5.83	1.59	2.37	27.97	3.90	10.63	13.23
kroD100	100	21294	16.64	9.67	4.08	3.48	21.35	7.33	12.25	12.18
kroE100	100	22068	16.72	9.35	3.57	1.23	21.07	7.32	9.33	9.92
pr76	76	108159	6.73	8.52	2.76	2.21	11.92	2.27	4.13	7.25
rat99	99	1211	18.23	7.84	2.96	2.26	85.40	1.98	8.67	7.72
rd100	100	7910	11.10	8.93	3.23	3.81	19.65	1.58	13.53	7.97
si175	175	21407	6.55	1.34	0.67	1.42	83.28	10.97	32.18	32.2
swiss42	42	1273	0.62	2.00	0	0	1.70	0.55	0.80	1.65
ulysses16	16	6859	0	0.16	0	0	0.27	0.42	0.13	0.16
ulysses22	22	7013	0	0.99	0	0	0.45	0.10	0.22	0.23
Average			5.96	4.66	1.43	1.18	14.91	2.75	5.66	11.08

Gap: the percentage difference between our solution and the optimal solution

Table 4.5: Computational results for TSPs less than or equal to 100 cities

Problem name	Size of problem	Optimal solution	EER gap (%)	SE gap (%)	ECER gap (%)	SP gap (%)	EER Time (mins)	SE Time (mins)	ECER Time (mins)	SP Time (mins)
a280	280	2579	51.39	15.36	10.88	11.37	553.23	43.10	83.05	126.40
brg180	180	1950	60.12	0	0.51	1.02	421	10.67	42.1	75.32
ch130	130	6110	15.37	8.07	3.99	6.13	57.45	4.48	17.30	24.18
ch150	150	6528	27.10	4.34	5.57	4.94	86.20	9.72	35.93	28.12
d198	198	15780	31.10	9.65	5.45	5.41	191.0	21.77	94.72	79.10
d493	493	35002	58.20	11.61	14.93	14.55	1947.28	216.13	503.32	845.9
eil101	101	629	17.99	8.71	4.55	4.12	35.35	14.35	14.27	11.72
fl417	417	11861	69.32	14.37	23.69	16.53	6506.23	183.83	727.90	478.32
gil262	262	2378	45.70	9.48	12.19	7.47	829.25	33.95	266.5	186.95
gr120	120	6942	21.62	13.53	8.04	6.21	38.78	3.32	19.18	15.35
gr137	137	69853	23.85	6.58	2.54	4.22	69.98	8.82	20.68	18.40
gr202	202	40160	30.65	12.83	6.99	6.87	186.60	21.95	45.38	79.27
gr229	229	134602	41.41	8.87	8.14	6.43	1186.98	40.42	69.10	124.22
gr431	431	171414	55.43	12.55	19.86	12.00	1351.13	129.95	1470.97	3225.55
kroA150	150	26524	29.15	7.10	8.01	5.17	93.55	16.98	31.38	42.2
kroB150	150	26130	30.46	9.96	8.77	6.55	83.08	6.10	30.3	27.85
kroA200	200	29368	40.97	10.46	7.72	5.91	225.37	21.57	60.78	51.27
kroB200	200	29437	37.61	11.99	6.75	8.86	262.25	84.43	68.90	48.23
lin105	105	14379	22.55	2.85	2.31	3.10	35.73	22.95	19.22	12.50
lin318	318	42029	53.18	12.06	12.49	12.99	729.87	59.82	502.13	386.98
linhp318	318	41345	54.62	11.76	17.45	12.33	830.35	50.35	212.43	261.03
pcb442	442	50778	60.29	14.11	19.80	12.76	2710.17	197.12	346.32	849.15
pr107	107	44303	20.46	5.79	2.11	2.98	44.52	2.63	10.15	7.65
pr124	124	59030	30.51	3.75	2.93	3.02	120.57	12.33	68.10	41.60
pr136	136	96772	26.50	12.54	12.43	6.19	110.02	51.63	25.20	33.77
pr144	144	58537	33.0	2.87	2.17	1.72	61.85	15.45	43.72	46.08
pr152	152	73682	31.14	4.99	4.98	1.87	66.35	12.68	31.53	20.68
pr226	226	80369	57.09	5.25	4.71	5.05	576.57	19.67	62.35	82.92
pr264	264	49135	54.08	10.26	10.49	9.37	1801.42	36.62	212.97	165.15
pr299	299	48191	55.19	15.96	13.04	16.80	1372.95	68.30	252.68	257.88
pr439	439	107217	62.42	17.29	20.57	13.45	2015.85	112.83	410.87	343.78
rat195	195	2323	33.70	9.65	5.72	4.17	245.22	12.88	77.12	64.48
rd400	400	15281	56.83	17.25	14.33	15.95	985.62	71.05	257.78	460.20
ts225	225	126643	46.34	6.91	6.04	3.27	214.75	22.18	36.73	56.58
tsp225	225	3919	42.49	8.26	10.30	10.40	272.12	14.23	107.91	58.05
u159	159	42080	37.87	11.90	2.83	2.91	70.5	8.95	26.63	20.83
Average			40.71	9.69	8.98	7.56	733.03	46.20	175.16	240.49

Gap: the percentage difference between our solution and the optimal solution

Table 4.6: Computational results for TSPs within 100 and 500 cities

Problem name	Size of problem	Optimal solution	EER gap (%)	SE gap (%)	ECER gap (%)	SP gap (%)	EER Time (mins)	SE Time (mins)	ECER Time (mins)	SP Time (mins)
att532	532	27686	67.58	17.72	16.18	18.41	3197.38	207.02	852.28	690.50
d657	657	48912	69.57	15.79	24.49	21.46	6132.67	328.17	1560.22	3255.72
gr666	666	294358	69.54	15.73	20.63	21.33	5501.32	474.17	1471.33	4465.62
p654	654	34643	86.92	20.92	34.21	21.69	7341.32	511.97	3200.60	2579.1
pa561	561	2763	65.41	15.30	20.14	16.09	3324.58	209.08	710.83	765.37
rat575	575	6773	67.22	16.69	18.42	16.34	4981.12	563.37	2119.12	1443.33
rat783	783	8806	72.88	19.89	25.96	23.28	10092.7	1265.88	2972.22	2763.28
si535	535	21407	66.85	56.68	57.94	57.17	3437.73	364.47	837.60	1064.08
u574	574	36905	68.33	15.67	21.50	20.82	4549.57	193.47	1536.10	1240.27
u724	724	41910	73.53	17.71	25.48	23.09	8500.60	1176.22	2849.62	2423.53
Average			70.78	21.21	26.50	23.97	5705.90	529.38	1810.99	2069.08

Gap: the percentage difference between our solution and the optimal solution

Table 4.7: Computational results for TSPs above 500 cities

Generally, our proposed operators produce better results than EER in quality of solutions obtained and CPU time used. Even for small-sized problems, our proposed operators outperform EER by about 4% (Table 4.5) and the CPU time used can be reduced to 9% of time used by EER. For SP, it produces the average result with gap of 1.18% whereas EER produces the result with gap of 5.95%. For SE, it produces better result than EER by using smaller amount of time. The performances of our proposed operators are more significant in solving large-sized problems. For large-sized problems (Table 4.7), the average gap obtained by using EER is about 70% whereas our proposed operators can obtain the solutions with gaps around 25%. Moreover, the CPU time used by EER is much longer than the time used by our proposed operators. It takes about 450% of the time taken by our proposed operators. It demonstrates the inability of EER in solving large-scale problems.

In most cases, SE operator produces a fair result given the smallest computational time. It converges faster than other operators did without losing the ability of finding a fair solution. For solving small and median-sized problems, SE is lagged behind SP and ECER. However, the difference diminishes when large problem instances are used.

The performance of ECER is in between SE and SP. It uses more time than SE and less time than SP. The results produced are also between SE and SP.

For solving small and median-sized problems (Table 4.5 and Table 4.6), SP produces the best results given the average gap is the smallest among all operators. However, the time used by SP is longer than SE and ECER. For solving large-sized problems (Table 4.7), SP is slightly lagged behind SE.

To conclude, using smaller production sizes and running time, our proposed operators still produce better solutions than other operators especially the EER. Note that this can save memory usage and CPU time. These two factors are very critical in solving large-scale TSPs. Also, our proposed operators have shown different contributions in solving the problem. When the user seek to obtain the best quality of solution, SP is a good choice. If time and quality are under concern, ECER is a better choice. On the other hand, SE is suitable when the user seek to have a fair solution in the shortest amount of time and is very effective in solving large-scale TSPs.

4.6 Conclusions

In this chapter, in order to enhance the performance of our new genetic based clustering algorithms, we propose three new GA operators. They are the Shortest Path (SP), the Shortest Edge (SE) and a modified version of the existing Enhanced Edge Recombination (EER) crossover operator, called Enhance Cost Edge Recombination (ECER) operator. The performances of our proposed operators are evaluated with seven existing operators using the TSPLIB problem instances, whose sizes range from 14 to 783 cities. Experimental results indicate that our proposed crossovers outperform other operators, especially the EER which is known to be a powerful crossover operator. By using fewer memory and lesser CPU time, our proposed operators are still able to produce better results than the others. Besides, they have different contributions in solving TSP: in most cases, SE performs the fastest, SP produces the best result and the performance of ECER is in between the two.

Chapter 5

Conclusions

5.1 Summary of Achievements

Clustering methods refer to a group of unsupervised pattern classification procedures that separate or partition a finite collection of objects into subsets to satisfy some predefined criteria. These methods have been used to solve many real-life problems. In this thesis, we examine the nature of a clustering problem and develop reliable and efficient algorithms.

Although many clustering algorithms have been proposed, their performance has not been extensively studied. Moreover, the special problem structure in clustering is rarely explored. Since the literature has shown that the clustering of a data array can be stated as the traveling salesman problem (TSP), the TSP structure may be useful in solving a clustering problem. In this thesis, we explore the use of genetic algorithms (GA) and propose a new GA clustering approach to solve the clustering problem. In particular, the TSP structure is exploited in our solution methodology. To demonstrate the use of our proposed algorithm, we consider several design problems in information systems. They are vertical partitioning (VP), horizontal partitioning (HP), object-oriented database (OODB) design and document database design.

Recently, OODB model has evolved as an alternative to the relational data model for supporting modern database applications. However, OODB presents additional semantics like structural properties (inheritance, composite objects) and interrelationships between objects. Hence, the existing clustering algorithms (used in relational databases, for instance) have to be adapted to the object-oriented model. In this thesis, we examine the use of our new GA clustering algorithms in designing an OODB.

Due to the rapid growth of the World Wide Web and hardware performance, a lot of database systems are built for storing documents. Moreover, the literature has shown that documents that stored in a database can be structurally organized. If similar documents based on its structure are partitioned together, the retrieval and query time of the database can be improved. In this thesis, we also demonstrate that our new clustering algorithm can be exploited to design structural document databases.

To evaluate the performance of our proposed algorithm, we compare our approach with the Slagle's, an efficient heuristic method for solving the VP problem. It should be noted that the Slagle's algorithm is similar to the bond energy algorithm (BEA) which is a well-known clustering algorithm. We generate several VP problems whose sizes range from 20 to 100 attributes. Computational results indicate that our proposed algorithm outperforms the Slagle's. It can further minimize the objective function and is able to produce more reasonable fragments.

In order to enhance the performance of the proposed algorithms, we propose three new GA operators for solving the TSP, namely Shortest Path (SP), Shortest Edge (SE) and a modified version of the existing Enhanced Edge Recombination (EER) crossover operator, called Enhance Cost Edge Recombination (ECER) operator. The performances of our proposed operators are evaluated with seven existing operators using the problem instances, whose sizes range from 14 to 783 cities. Experimental results indicate that our proposed crossover operators outperform others, especially the EER operator which is known to be very powerful. By using fewer memory and

lesser CPU time, our proposed operators are still able to produce better results than the others. Besides, they have different contributions in solving TSP: in most cases, SE performs the fastest operation, SP produces the best result and the performance of ECER is in between the two.

5.2 Future Development

In distributed/parallel database system, low level design such as hardware parameters can be incorporated into the cost function so as to produce more precise fragments. Also, after data are partitioned, we can allocate them to different sites/machines using the proposed algorithm. Besides, if the cost function is carefully modified, the proposed algorithm can be used in distributed query processing to allocate operations of queries to different sites. The application can be extended in designing mixed partitioning, overlapping fragments and fragments used in different memory level.

Apart from information systems, our clustering algorithm can be applied to many different areas whenever we can formulate a proper cost function for the underlying application. For instance, we can use it in re-ordering the part-machine matrix for cellular manufacturing. This problem is concerned with the identification of machines to be included in each cell as well as the specification of the cells where each part is to be processed. We can define the cost measure for machines and parts based on the part-machine incidence matrix. The objective is to cluster machines/parts into groups such that machines/parts located within the same group have lower costs whereas those between groups have higher costs.

For GA, we can analysis the choice of mutation operators in solving the TSP. Different mutation operators and other coding methods can be developed to further enhance the system. Besides, we can develop hybrid crossover that exploit the benefits of different operators.

- END -

Bibliography

[1] R. G. Askin, S. H. Cresswell, J. B. Goldberg and A. J. Vakharia, "A Hamiltonian Path Approach to Reordering the Part-machine Matrix for Cellular Manufacturing", *Int. J. Prod. Res.*, Vol. 29, No. 6, pp. 1081-1100, 1991.

[2] M. Abdelguerfi and K .F. Wong, *Parallel Database Techniques*, IEEE-CS Press, 1998.

[3] M. R. Anderberg, *Cluster Analysis for applications*, Academic Press, New York, 1973.

[4] J. E. Baker, "Reducing Bias and Inefficiency in the Selection Algorithm", in J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 14-21, Lawrence Erlbaum Associates, 1987.

[5] C. K. Bayne, J. J. Beauchamp, C. L. Begovich, and V. E. Kane, "Monte Carlo Comparisons of Selected Clutering Procedures", *Pattern Recognition*, 12, pp. 51-62, 1980.

[6] David Beasley, David R. Bull and Ralph R. Martin, "An Overview of Genetic Algorithms: Part 1, Fundamentals", *University Computing*, 15(2), pp. 58-69, 1993.

[7] L. Bellatreche, A. Simonet and M. Simonet, "An Algorithm for Vertical Fragmentation in Distributed Object Database Systems with Complex Attributes and Methods", *in 7th Intl. Conf. on Databases and Expert Systems*, DEXA96, pp. 15-21, IEEE Computer Society, Zurich, Switzerland, September, 1996.

[8] B. E. Bellman, "Dynamic Programming Treatment of the Traveling Salesman Problem", *Journal of the Association for Computing Machinery*, 9, pp. 61-63, 1963.

[9] J. N. Bhuyan, V. V. Raghavan, V. K. Elayavalli, "Genetic Algorithm for Clustering with an Ordered Representation", *Proceedings of the fourth International Conference on Genetic Algorithms*, in R. K. Belew, L. B. Booker, editor, Morgan Kaufmann, pp. 408-415, 1991.

[10] R. G. Bland and D. F. Shallcross, "Large Traveling Salesman Problems Arising From Experiments in X-Ray Crystallography: A Preliminary Report in Computation", *Operations Research Letters*, Vol. 8, pp. 125-128, 1989.

[11] H. Braun, "On Traveling Salesman Problems by Genetic Algorithms", *1st Workshop on Parallel Problem Solving from Nature*, pp. 129-133, Oct., 1990.

[12] S. Ceri, and G. Pelagate, *Distributed Databases: Principle and Systems*, McGraw-Hill, New York, 1984.

[13] C. H. Cheng, "A Branch and Bound Clustering Algorithm", *IEEE Transactions on Systems*, Man, and Cybernetics, SMC-25, pp. 895-898, 1995.

[14] C. H. Cheng, Y. P. Gupta, W. H. Lee and K. F. Wong, "A TSP-based Heuristic for Forming Machine Groups and Part Families", *Int. J. Prod. Res.*, Vol. 36, No. 5, pp. 1325-1337, 1998.

[15] Gajanan S. Chinchwadkar, Angela Goh and Ee-Peng Lim, "Simulated Annealing for Vertically Partitioning an OO Database", *International Conference on Information, Communications and Signal Processing*, ICICS '97 Singapore, 9-12 September, pp. 800-804, 1997.

[16] N. Christofides, "Worst-case Analysis of a New Heuristic for The Traveling Salesman Problem", Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pa, 1976.

[17] Hong K. Chung and John P. Norback, "A Clustering and Insertion Heuristic Applied to a Large Routing Problem in Food Distribution", *J. Op. Res. Soc.*, Vol. 42, No.7, pp. 555-564, 1991.

[18] D. W. Cornell and P. S. Yu, "A Vertical Partitioning Algorithm for Relational Databases", *Proceedings of the Third International Conference on Data Engineering*, pp. 30-35, 1987.

[19] D. W. Cornell and P. S. Yu, "An Effective Approach to Vertical Partitioning for Physical Design of Relational Databases", *IEEE Trans. Software Eng.*, 16(2), pp. 248-258, 1990.

[20] C. J. Date, *An Introduction to Database Systems*, 6th edition, Addison-Wesley, Massachusetts, 1995.

[21] L. Davis, "Applying Adaptive Algorithms to Epistatic Domains", *Proc. 9th International Joint Conference on Artifical Intelligence*, pp. 162-164, 1985.

[22] L. Davis, *Genetic Algorithms and Simulated Annealing*, Pitman, 1987.

[23] L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[24] K. DeJong, *The Analysis and Behaviour of a Class of Genetic Adaptive Systems*, Ph.D. thesis, University of Michigan, 1975.

[25] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", *Numerische Mathematik*, 1, pp. 269-271, 1959.

[26] R. C. Dixon, "Lore of the Token Ring", *IEEE Network Magazine*, Vol. 1, pp. 11-18, Jan. 1987.

[27] A. W. F. Edwards and L. L. Cavalli-Sforza, "A Method for Cluster Analysis", *Biometrics*, 21, pp. 362-375, 1965.

[28] M. J. Eisner and D. G. Severance, "Mathematical Techniques for Efficient Record Segmentation In Large Shared Databases", *Journal of ACM*, 23, pp. 619-635, 1976.

[29] C. I. Ezeife and K. Barker, "Vertical Class Fragmentation in a Distributed Object Based System", Technical Reprot: 94-03, Dept. of Computer Science, Uni. Of Manitoba, Canada, 1994.

[30] E. Falkenauer, *Genetic Algorithms and Grouping Problems*, John Wiley & Sons, 1998.

[31] D. B. Fogel, L. J. Fogel, and J. W. Atmar., "Meta-evolutonary Programming", in R. R. Chen, editor, *Proceedings of the 25th Asilomar Conference on Signals, Systems and Computers*, pp. 540-545, San Jose, CA, Maple Press, 1991.

[32] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[33] D. E. Goldberg, "Computer-aided Gas Pipeline Operation Using Genetic Algorithms and Rule Learning", (Doctoral dissertation, University of Michigan),

Dissertation Abstract International, 44(10), 3174B, (University Microfilms No. 8402282), 1983.

[34] D. Goldberg and R. Lingle, "Alleles, Loci, and the Traveling Salesman Problem", *First International Conference on Genetic Algorithms and Their Applications*, pp. 154-159, 1985.

[35] D. E. Goldberg and R. E. Smith, "Blind Inferential Search with Genetic Algorithms", paper presented at ORSA/TIMS Joint National Meeting, Miami, FL, 1986.

[36] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, Reading, MA, 1989.

[37] R. R. Golden, P. E. Meehl, "Detection of Biological Sex - An Empirical Test of Cluster Methods", *Multivariate Behavioral Research*, 15, pp. 475-496, 1980.

[38] R. L. Graham *et al.*, *Handbook of Combinatorics*, Vol. 2, The MIT Press, north-Holland, 1995.

[39] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Gucht, "Genetic Algorithms for the Traveling Salesman Problem", *First International Conference on Genetic Algorithms and Their Applications*, pp. 160-168, 1985.

[40] J. J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms", *IEEE Trans. SMC*, 16, pp. 122-128, 1986.

[41] J. Grefenstette, "Incorporating Problem Specific Knowledge in Genetic Algorithms", *Genetic Algorithms and Simulated Annealing*, Lawrence Davis, editor, pp. 42-60, 1987.

[42] J. J. Grefenstette, "Genetic Algorithms and their Applications", In A. Kent and J. G. Williams, editors, *Encyclopaedia of Computer Science and Technology*, pp. 139-152, Marcel Dekker, 1990.

[43] John A. Hartigan, *Clustering Algorithms*, John Wiley & Sons, 1975.

[44] John A. Hartigan, "Statistical Theory in Clustering", *Journal of Classification*, 2, pp. 63-76, 1985.

[45] M. Hammer and B. Niamir, "A Heuristic Approach to Attribute Partitioning", *ACM SIGMOD International Conference on Management of Data*, pp. 93-101, 1979.

[46] R. L. Haupt, S. E. Haupt, *Practical Genetic Algorithms*, John Wiley & Sons, 1998.

[47] A. Hertz and D. de Werra, "Using Tabu Search Techniques for Graph Coloring", *Computing*, 29, pp. 345-351, 1987.

[48] F. S. Hillerand and G. J. Lieberman, *Introduction to Operations Research*, Holden-Day, California, 1986.

[49] J. A. Hoffer, "An Integer Programming Formulation of Computer Database Design Problems", *Information Sciences*, 11, pp. 29-48, 1976.

[50] J. H. Holland, *Adaptation in Natural and Artificial Systems*, MIT Press, 1975.

[51] A. Homaifar, S. Guan, and G. Liepins, "A New Approach on the Traveling Salesman Problem", *Fifth International Conference on Genetic Algorithms*, pp. 460-466, July, 1993.

[52] Yin-Fu Huang, Chin-Huei Van, "Vertical Partitioning in Database Design", *Information Sciences*, 86, pp. 19-35, 1995

[53] L. J. Hubert and J. Schultz, "Quadratic Assignment as a General Data-Analysis Strategy", *British Journal of Mathematical and Statistical Psychology*, 29, pp. 190-241, 1976.

[54] R. Dubes and A. K. Jain., "Clustering Techniques: the User Dilemma", *Pattern Recognition*, 8, pp. 247-268, 1980.

[55] R. Dubes and A. K. Jain., "Clustering Methodologies in Exploratory Data Analysis", *Adv. Comput.*, 19, pp. 213-228, 1980.

[56] A. K. Jain, R. C. Dubes, *Algorithms for Clustering Data*, Prentice Hall, New Jersey, 1988.

[57] R. E. Jensen, "A Dynamic Programming Algorithm for Cluster Analysis", *Operations Research*, 17, pp. 1034-1057, 1969.

[58] M. F. Jiang, S. S. Tseng, C. J. Tsai, "Intelligent Query Agent for Structural Document Databases", *Expert Systems with Applications*, 17, pp. 105-113, 1999.

[59] M. F. Jiang, S. S. Tseng, C. J. Tsai, "Discovering Structure from Document Databases", *The Third Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 169-173, PAKDD-99, Beijing, China.

[60] P. Jog, J. Suh and D. Gucht, "The Effect of Population Size, Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem", *Third International Conference on Genetic Algorithms*, pp. 110-115, June, 1989.

[61] D. S. Johnson, "Local Optimization and the Traveling Salesman Problem", in M.S. Paterson (Editor), *Proceedings of the 17th Colloquium on Automata, Languages, and Programming*, Springer-Verlag, Lecture Notes in Computer Science, Vol. 443, pp. 446-461, 1990.

[62] S. Kirkpatrick, C. D. Gelatt and M. P.Vecchi, "Optimization by Simulated Annealing", *Science*, 220, pp. 671-680, 1983.

[63] J. Knox, "The Application of TABU Search to the Symmetric Traveling Salesmane Problem", Ph.D. dissertation, University of Colorado, 1989.

[64] B. Korte, "Applications of Combinatiorial Optimization", talk at the 13th International Mathematical Programming Symposium, Tokyo, 1988.

[65] Warren L. G. Koontz, P. M. Narendra and K. Fukunaga, "A Branch and Bound Clustering Algorithm", *IEEE Trans. Comput.*, pp. 908-915, 1975.

[66] A. Kusiak and W.S. Chow, "An Algorithm for Cluster Identification", *IEEE Transactions on Systems*, Man, and Cybernetics, SMC-17, pp. 696-699, 1987.

[67] J. Lam, "An Efficient Simulated Annealing Schedule", Ph.D. dissertation, Department of Computer Science, Yale University, 1988.

[68] E. L. Lawler, et al., *The Traveling Salesman Problem*, Wiley-Interscience Publication, 1985.

[69] E. L. Lawler and D.E. Wood, "A Branch-and-Bound Methods", *A Survey, Operations Research*, Vol. 14, pp. 699-719, 1966.

[70] L. P. Lefkovitch, "Conditional Clustering", *Biometrics*, 36, pp. 43-58, 1980.

[71] J. K. Lenstra, "Clustering a Data Array and The Traveling Salesman Problem", *Oper. Res.*, Vol. 22, pp. 413-414, 1974.

[72] J. K. Lenstra, and A. H. G. Kan Rinnooy, "Some Simple Applications of The Traveling Salesman Problem, *Operations Research Quarterly*, 26, pp. 717-733, 1975.

[73] T. Liang, "The Study of Character-based Signature Methods in Chinese Text Retrieval", Ph.D. thesis, National Chiao Tung University, Taiwan, 1995.

[74] S. Lin, and B. W. Kernighan, "An Effective Heuristic Algorithm for Traveling Salesman Problem", *Operation Research*, pp. 498-516,1973.

[75] X. Lin, M. Orlowska and Y. Zhang, "A Graph Based Cluster Approach for Vertical Partitioning in Database Design", *Data and Knowledge Engineering*, Vol. 11, No. 2, pp. 151-169, 1993.

[76] X. Lin and Y. Zhang., "A New Graphical Method for Vertical Partitioning in Distributed Database Design", *Proceedings of The Fourth Australian Database Conference*, pp. 131-144, 1993.

[77] J. D. Litke, "An Improved Solution to the Traveling Salesman Problem with Thousands of Nodes", *Communications of the ACM*, Vol. 27, No. 12, pp. 1227-1236, 1984.

[78] Chiun-Ming Liu, "Clustering Techniques for Stock Location and Order-picking in a Distribution Center", *Computer & Operations Research*, 26, pp. 989-1002, 1999.

[79] O. Martin, S. W. Otto and E. W. Felten, "Large Step Markov Chains for the Traveling Salesman Problem", *Complex Systems*, 2, pp. 299-326, 1991.

[80] W. T. McCormick , P. J. Schwietzer and T. W. White, "Problem Decomposition and Data Reorganization by A Clustering Technique", *Operat. Res*, 20, pp. 993-1009, 1972.

[81] P. Miliotis, "Integer Programming Approaches to the Traveling Salesman Problem", *Math. Progr.*, 10, pp. 367-378, 1976.

[82] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, (Third, Revised and Extended Edition), Hong Kong: Springer, 1999.

[83] G. W. Milligan, "An Examination of the Effect of Six Types of Error Perturbation on Fifteen Clustering Algorithms", *Psychometrika*, 45, pp. 325-342, 1980.

[84] G. W. Milligan, S. C. Soon and L. M. Sokol, "The Effect of Cluster Size Dimensionality, and the Number of clusters on Recovery of True Cluster Structure", *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI*, 5, pp. 40-47, 1983.

[85] G. W. Milligan and M. C. Cooper, "An Examination of Procedures for Determining the Number of Clusters in a Data Set", *Psychometrika*, 50, pp. 159-179, 1985.

[86] G. Navarro & R. Baeza-Yates, "Proximal Nodes: A Model to Query Document Database by Content and Structure", *ACM Transactions on Information Systems*, 15 (4), pp. 400-435, 1997.

[87] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou, "Vertical Partitioning Algorithm for Database Design", *ACM Transactions on Database Systems*, 9, pp. 680-710, 1984.

[88] S. Navathe and M. Ra, "Vertical Partitioning for Database Design: A Graphical Algorithm", *Proceedings of ACM SIGMOD*, pp. 367-378, 1989.

[89] Hans-Thomas Nürnberg and Hans-Georg Beyer, "Optimization of Traveling Salesman Problem", in Angeline *et al.*, editor, *Lecture Notes in Computer Science*, 1213, Evolutionary Programming VI, pp. 349-359, Springer, 1997.

[90] M. Ozsu, and P. Valduriez, *Principles of Distributed Database Systems*, Prentice Hall, New Jersey, 1991.

[91] Oliver, I.M., Smith, D.J., and Holland, J.R.C., "A Study of Permutation Crossover Operators on the Traveling Salesman Problem", *Proceedings of the Second International Conference on Genetic Algorithms*, San Mateo, California (CA: Morgan Kaufmann), pp. 224-230, 1987.

[92] M. Padberg and G. Rinaldi, "Optimization of a 532-city Symmetric Traveling Salesman Problem by Branch and Cut", *Operation Res. Letter*, 6, pp. 1-7, 1987.

[93] J. Puzicha, T. Hofmann, J. M. Buhmann, "Histogram Clustering for Unsupervised Segmentation and Image Retrieval", *Pattern Recognition Letters*, 20, pp. 899-909, 1999.

[94] M. R. Rao, "Cluster Analysis and Mathematical Programming", *Journal of the American Statistical Association*, 66, pp. 622-626, 1971.

[95] G. Reinelt, "TSPLIB - A Traveling Salesman Problem Library", *ORSA Journal of Computing*, 3, pp. 376-384, 1991.
(ftp://ftp.zib.de/pub/mp-testdata/tsp/tsplib/index.html)

[96] G. Reinelt, *The Traveling Salesman: Computational Solutions for TSP Applications, Lecture Notes in Computer Science*, Vol. 840, Springer Heidelberg, 1994.

[97] M. B. Rosenwein, "An Application of Clustering Analysis to the Problem of Locating Items within a Warehouse", *IIE Transactions*, Vol. 26, No. 1, pp. 101-103, 1994.

[98] D. Rosenkrantz, R. Stearns and P. Lewis, "Approximate Algorithms for the Traveling Salesperson Problem", *Proceedings of the 15th Annual IEEE Symposium of Switching and Automata Theory*, pp. 33-42, 1974.

[99] F. E. Ross, "FDDI - A Tutorial", *IEEE Commun. Magazine*, Vol. 24, pp. 10-15, 1986.

[100] J. R. Slagle, C. L. Chang, and S. R. Heller, "A Clustering and Data-reorganization Algorithm,", *IEEE Trans. Syst.*, Man. Cybern., Vol. SMC-5, pp. 125-128, Jan. 1975.

[101] R. Sproat and C. Shilh, "A Statistical Method for Finding Word Boundaries in Chinese Text", *Computer Proceedings of Chinese and Oriental Languages*, 4 (4), pp. 336-351, 1990.

[102] L. E. Stanfel, "Application of Clustering to Information System Design", *Inform. Processing & Management*, Vol. 19, pp. 37-50, 1983.

[103] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley and C. Whitley, "A Comparison of Genetic Sequencing Operators", *Proceedings of the fourth International Conference of GAs and their applications*, pp. 69-76, 1991.

[104] G. Syswerda, "Uniform Crossover in Genetic Algorithms", in J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms"*, pp. 2-9, Morgan Kaufmann, 1989.

[105] G. Syswerda, "Schedule Optimization Using Genetic Algorithms", in L. Davis, editor, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, chapter 21, pp. 332-349, 1990.

[106] H. Tamaki *et al.*, "A Comparison Study of Genetic Codings for the Traveling Salesman Problem", *Proceeding of The First IEEE Conference On Evolutionary Computation*, Vol. 1, pp. 1-6, 1994.

[107] J. Tou and R. Gonzalez, *Pattern Recognition*, Reading, M.A.: Addison-Wesley, 1974.

[108] N. Ulder, E. Aarts, H. Banbelt, P. Laahoven, and E. Pesch, "Genetic Local Search Algorithms for Traveling Salesman Problem", *1st Workshop on Parallel Problem Solving from Nature*, pp. 109-116, Oct., 1990.

[109] H. D. Vinod, "Integer Programming and Theory of Grouping", *Journal of The American Statistical Association*, 64, pp. 506-519, 1969.

[110] D. Whitley, "Using Reproductive Evaluation to Improve Genetic Search and Heuristic Discovery", in J. J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms"*, pp. 108-115, Lawrence Erlbaum Associates, 1987.

[111] D. Whitley, "The Genitor Algorithm and Selection Pressure: Why Rank-based Allocation of Reproductive Trials Is Best", *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, California (CA: Morgan Kaufmann), pp. 116-121, 1989.

[112] D. Whitley, T. Starkweather and D. A. Fuquay, "Scheduling Problems and Traveling Salesman: the Genetic Edge Recombination Operator", *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, Californai (CA: Morgan Kaufmann), pp. 133-140, 1989.

[113] G. Wiederhold, *Database Design*, McGraw-Hill, New York, 1982.

[114] Y. Zhang, and M. E. Orlowska, "On Fragmentation Approaches for Distributed Database Design", *Information Sciences*, 1, pp. 117-132, 1994.

[115] S. Zhou, H. M. Williams and K. F. Wong, "Data Placement in Shared-Nothing Database Systems", *High Performance Cluster Computing*, Vol. 2, Prentice Hall, pp. 440-453, 1999.